VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELOVÁNÍ KVALITY SLUŽEB V POČÍTAČOVÝCH SÍTÍCH

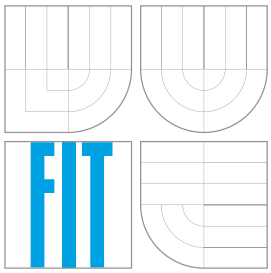
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN DANKO

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELOVÁNÍ KVALITY SLUŽEB V POČÍTAČOVÝCH SÍTÍCH

MODELLING QOS IN COMPUTER NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN DANKO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ONDŘEJ RYŠAVÝ, Ph.D.

BRNO 2012

Abstrakt

Kvalita služeb (QoS) je důležitým aspektem v oblasti počítačových sítí, ale je také velkou výzvou. Poskytnutí záruky QoS je ještě náročnější, když přidáte složitosti do sítě, jako jsou hlasové a video aplikace. Tato diplomová práce se zaměřuje na modelování a simulaci QoS v diskrétním simulačním systému OMNeT++. Implementace několika technik pro simulaci QoS je přidána do OMNeT++ v rámci této práce. Všechny implementované moduly jsou ověřené vůči QoS nástrojům na reálných směrovačích. Poslední část práce představuje možnosti simulace QoS v nově implementovaných modulech.

Abstract

Quality of service (QoS) is an important consideration in networking, but it is also a significant challenge. Providing QoS guarantees becomes even more challenging when you add the complexities to the network like voice and video applications. This master's thesis focuses on QoS modeling and simulation in discrete event simulation system OMNeT++. The implementation of multiple techniques for QoS simulation is added to OMNeT++ within this work. All implemented modules are validated against the QoS tools on real routers. The last part of the work presents the possibility of QoS simulation in newly implemented modules.

Klíčová slova

Kvalita služeb, QoS, Diferencované služby, OMNeT++, INET Framework, diskrétní simulace

Keywords

Quality of Services, QoS, Differentiated services, OMNeT++, INET Framework, discrete simulation

Citace

Martin Danko: Modelování kvality služeb v počítačových sítích, diplomová práce, Brno, FIT VUT v Brně, 2012

Modelování kvality služeb v počítačových sítích

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Ondřeje Ryšavého Ph.D.

.....
Martin Danko
10. mája 2012

Poděkování

Děkuji vedoucímu mé diplomové práce Ing. Ondřejovi Ryšavému Ph.D. za možnost odborných konzultací a za jeho cenné rady.

Obsah

Kapitola 1

Úvod

Trendom pri dizajnovaní dnešných moderných počítačových sietí je vytvorenie jednej konvergovanej infraštruktúry pre prenos rôznych typov služieb. Dochádza k integrácii hlasových a video služieb do infraštruktúry určenej primárne pre prenos dát. Najmodernejšie sieťové prvky dokonca umožňujú integrovať služby SAN do segmentu LAN sietí.

Táto integrácia so sebou prináša nutnosť rozlišovať jednotlivé typy služieb a prideľovať im patričnú prioritu v rámci prenosu sieťovou infraštruktúrou. V anglickej terminológií je pre túto schopnosť siete zažitý výraz Quality of Service (QoS). QoS znamená schopnosť poskytovať rôznu prioritu rôznym aplikáciám, užívateľom alebo dátovým tokom. Taktiež umožňuje garantovať konkrétne prenosové parametre jednotlivým dátovým tokom. Napríklad môže byť garantovaná potrebná prenosová rýchlosť, oneskorenie, rozptyl oneskorenia a stratovosť paketov.

Obsahom tejto práce je analýza možností simulácie QoS v nástroji OMNeT++ v prostredí INET Framework. Podľa výsledkov analýzy budú implementované vybrané chýbajúce nástroje pre simuláciu QoS tak, aby simulácia čo najviac zodpovedala možnostiam na reálnych sieťových prvkoch. Funkčnosť implementovaných nástrojov bude overovaná voči ich správaniu sa na hardwarových smerovačoch Cisco ISR 2811.

Práca je vytvorená v rámci výskumnej skupiny NES@FIT a je súčasťou projektu ANSA (Automated Network-Wide Security Analysis)[8]. Projekt sa zaoberá verifikáciu sieťových topológii na špecifické vlastnosti ako dostupnosť, bezpečnosť či schopnosť odolávať poruchám. Práca bude spolu s prácami ostaných študentov vytvárať modul reálneho smerovača v prostredí INET Framework. Pomocou tohto smerovača bude následne možné vytvárať a simulovať komplexné topológie rozsiahlych sieti.

Kapitola 2

Techniky pre zaistenie QoS

Táto kapitola je úvodom do problematiky kvality služieb (QoS) a obsahuje prehľad typických nástrojov používaných pre zabezpečenie QoS v produkčných sieťach.

Dôležitosť tejto kapitoly spočíva v definovaní a objasnení pojmov z oblasti QoS, ktoré budú následne diskutované v ďalších kapitolách.

2.1 Úvod do QoS

Typická definícia QoS, ktorú nájdeme v mnohých edukačných materiáloch znie nasledovne[6]: *"QoS je schopnosť siete zabezpečovať lepšie alebo špeciálne prenosové služby pre skupinu užívateľov alebo aplikácie alebo pre obe súčasne na úkor iných užívateľov alebo aplikácií."*

Otázka potreby QoS je spôsobená konečnosťou množstva prostriedkov počítačovej siete. Typicky dochádza k nasledovným súpereniam o prostriedky:

- Súperenie užívateľov – nastáva, keď pakety rôznych užívateľov, užívateľských skupín, respektíve oddelení, súťažia o rovnaké limitované prostriedky siete.
- Súperenie aplikácii – nastáva, keď rôzne aplikácie súťažia o rovnaké limitované prostriedky siete. Tieto aplikácie majú typicky rôzne prenosové nároky.

Všetky komponenty siete ovplyvňujú úroveň QoS poskytovaný danou sieťou. Preto by na celej ceste on zdroja až ku cieľu, kde je potrebné zabezpečiť určitú úroveň QoS, mala byť na prvkoch podporovaná a nakonfigurovaná potrebná úroveň QoS služieb. Je nesporné, že výkon celej siete výrazne závisí na najslabšom prvku v sieti. Jedná sa o aktívne prvky, u ktorých je možné ovplyvňovať zachádzanie s jednotlivými paketmi. Na pasívnych prvkoch ako sú fyzické linky, ktoré žiadnu konfiguráciu neobsahujú, nie je možné parametre QoS priamo ovplyvňovať . Vysoká úroveň QoS sa často spája s vysokou úrovňou výkonu siete a dosiahnutými prenosovými parametrami, ako napríklad vysokou prenosovou rýchlosťou, nízkym oneskorením, nízkou pravdepodobnosťou straty paketu.

QoS môže zlepšiť prenosové parametre pre kritické dáta a aplikácie, obmedziť prenosy menej dôležitým sieťovým prenosom a poskytnúť konzistentné správanie siete. Pomocou mechanizmov QoS je možné zvýšiť efektívnosť drahých sieťových pripojení. Dokáže zabrániť vyčerpávaniu sieťových zdrojov aplikáciám, ktoré nie sú pre vlastníka siete kritické, vďaka čomu je možné zvýšiť produktivitu.

2.2 Hlavné parametre QoS

QoS sa určuje meraním dostupnosti služby a prenosovej kvality. Dostupnosť je rozhodujúcim základným elementom pre každú implementáciu QoS. Pred tým ako môže byť nejaká implementácia QoS úspešná, musí byť sieťová infraštruktúra dizajnovaná tak, aby bola odolná voči výpadkom. Cieľová dostupnosť produkčných sieti sa pohybuje na úrovni 99,999 percent, čo znamená 5 minút povoleného výpadku siete za rok. Hlavnými parametrami prenosovej kvality sú:

- Šírka pásma – Bandwith [bit/s]
- Oneskorenie – Delay [ms]
- Rozptyl oneskorenia – Jitter [ms]
- Stratovosť paketov – Packet loss [%]

2.2.1 Šírka pásma

Termín šírka pásma udáva počet bitov za sekundu, ktoré je možné úspešne preniesť daným médiom. Dostupná šírka pásma je značne závislá od fyzikálnych vlastnosti siete a použitých technológií. Maximálna šírka pásma pre komunikáciu od daného zdroja k danému cieľu je určená šírkou pásma linky s najnižšou hodnotou pozdĺž cesty paketov. Priemerná dostupná šírka pásma je určená maximálnou šírkou pásma rozdelenou počtom tokov.

Nedostatočná šírka pásma spôsobuje oneskorenie, stratu paketov a nízky výkon aplikácií. Riešiť nedostatok šírky pásma je možné jedným z nasledovných spôsobov:

- Zvýšením šírky pásma na linke – efektívne ale nákladné riešenie.
- Využitie techniky prioritizácie na výstupných frontách – posielať dôležité pakety ako prvé.
- Použitie kompresie prenášaných dát – kompresia na úrovni L2, kompresia TCP hlavičiek alebo použitie cRTP.

Zvýšenie šírky pásma je nepochybne prínosné, ale tento proces nemusí byť jednoduchý a vykonaný časovo dostatočne rýchlo. Hlavným nežiaducim dôsledkom sú najmä zvýšené náklady. Samotné zvýšenie šírky pásma nemusí byť stále efektívne, problémy sa môžu objaviť pri nárazovom nahromadení prenášaných dát. V niektorých prípadoch by malo byť zvýšenie šírky pásma linky prvou vykonanou akciu, ale nemala by byť jedinou.

2.2.2 Oneskorenie

Celkové oneskorenie alebo tiež jednocestné oneskorenie je čas, za ktorý trvá prenos paketu z vysielacieho koncového zariadenia do cieľového koncového zariadenia. Celkové oneskorenie je súčtom nasledovných čiastkových oneskorení:

- Oneskorenie spôsobené spracovaním – je doba, ktorú trvá sieťovým prvkom ako smerovače alebo prepínače vykonať všetky potrebné operácie k preneseniu paketu z vstupného rozhrania na výstupné rozhranie. Typ CPU, vyťaženie CPU, technológia prepínania paketov, architektúra prvku a nakonfigurované funkcie na zariadení ovplyvňujú tento typ oneskorenia.

- Oneskorenie spôsobené radením do front – je doba, ktorú paket strávi vo výstupnej fronte na rozhraní sieťového zariadenia. Vyťaženie zariadenia, množstvo paketov čakajúcich vo fronte, spôsob plánovania a šírka pásma linky ovplyvňujú tento typ oneskorenia.
- Serializačné oneskorenie – je doba, ktorú trvá vyslanie všetkých bitov dátového rámca na fyzické médium k prenosu. Pokiaľ je linka rýchla, bity môžu byť na linku vyslané omnoho rýchlejšie ako v prípade pomalej linky. Takisto, v prípade, že je dátový rámec krátky, bity môžu byť na linku vyslané omnoho rýchlejšie ako v prípade dlhého rámca.
- Propagačné oneskorenie – je doba, ktorú trvá prenos bitu z jedného konca linky na druhý. V prípade, že je vyslaný elektrický alebo optický signál do média, energia sa nespropaguje na druhú stranu média okamžite. Rýchlosť energie či už elektrickej alebo optickej sa približuje rýchlosti svetla a túto rýchlosť nie je možné zmeniť. Jediná premenná, ktorou je možné meniť propagačné oneskorenie je dĺžka linky.

2.2.3 Rozptyl oneskorenia

Pri prenose dát počítačovou sieťou sú jednotlivé pakety spracovávané nezávisle od tých ostatných a môžu sieťou putovať rôzne dlho podľa dĺžky zvolenej cesty alebo jej zahltenia. Môže nastať aj situácia, keď sú pakety prijaté na cieľovej stanici v opačnom poradí ako boli odoslané zo zdrojovej stanice. Takéto premenlivé oneskorenie medzi doručením po sebe nasledujúcich paketov sa nazýva jitter – rozptyl (variácia) oneskorenia. V ideálnom prípade by mal príjemca prijímať pakety s rovnakým intervalom ako boli odoslané. Hodnota rozptylu oneskorenia by bola nulová.

Pre potlačenie rozptylu oneskorenia sa používa vyrovnávacia pamäť nachádzajúca sa na cieľovej stanici medzi sieťovou vrstvou a cieľovou aplikáciou. Inštalovaná vyrovnávacia pamäť sa snaží potlačiť kolísavosť doby príchodu paketov tým, že pozdrží pakety na určitú dobu a následne posiela aplikácii pakety v správnom poradí s pôvodným intervalom medzi jednotlivými paketami. Použitím vyrovnávacej pamäte ale dochádza k celkovému zvýšeniu jednosmerného oneskorenia v rámci danej komunikácie.

2.2.4 Stratovosť paketov

Stratovosť paketov vyjadruje pomer paketov, ktoré nedorazia k cieľovej aplikácii, k celkovému počtu prenášaných paketov. K strate dochádza v dôsledku dočasného preťaženia alebo nedostupnosti niektorého sieťového zariadenia alebo nedostupnosti niektorej z liniek na komunikačnej trase. Typicky môže dôjsť k preťaženiu procesora smerovača, dočasnému výpadku linky a následnej konvergencii sieti, zahlteniu výstupnej fronty rozhrania smerovača, kolízii na linkovej vrstve alebo externému rušeniu na použitom médiu.

Dopad straty paketu závisí na použitom transportnom protokole a samotnej aplikácii. V prípade použitia prenosového protokolu TCP je stratený paket znovu prenesený v rámci mechanizmu spoľahlivého prenosu samotného protokolu. Vedľajším efektom je zníženie veľkosti okna (window size) a tým aj zníženiu prenosovej rýchlosti. Pokiaľ dôjde k strate paketu v prenose pomocou transportného protokolu UDP, tak sa o znovu zaslanie musí postarať samotná aplikácia. V závislosti na implementácii konkrétnej aplikácie môže kvôli strate jedného paketu dochádzať k nutnosti znovu zaslania celého súboru. U niektorých aplikácií je z princípu nezmyslel znovu zasielať stratené pakety. Jedná sa hlavne o real-time aplikácie hlasových alebo video služieb. U tohto typu aplikácií je nutné zaistiť čo najmenšiu stratovosť.

2.3 Modely QoS

V nasledujúcej časti budú diskutované tri hlavné modely QoS vyskytujúce sa v IP sieťach. Konkrétne ide o model najlepšieho úsilia (best-effort), model integrovaných služieb (IntServ) a model diferencovaných služieb (DiffServ). Vždy budú zhrnuté kľúčové výhody a tiež nedostatky všetkých týchto modelov QoS.

2.3.1 Best-effort model

Siete bez implementovaného QoS sú charakterizované ako tzv. siete najlepšieho úsilia (best-effort). Z pohľadu dôležitosti paketov sú v sieti modelu best-effort všetky pakety vnímanie ako rovnako dôležité. Takéto siete fungujú dobre pokiaľ je zaistené dostatočné množstvo prostriedkov (CPU, pamäť, šírka pásma) pre okamžité spracovanie všetkých paketov prechádzajúcich sieťou.

Best-effort model má niekoľko svojich výhod a tak isto pár nedostatkov. Výhody modelu best-effort sú nasledovné:

- Škálovateľnosť – celosvetová sieť Internet je založená na modelu best-effort. Teda tento model nemá prakticky žiadne limity z hľadiska škálovateľnosti. Šírka pásma na linkách prepájajúcich jednotlivé sieťové prvky určuje efektivitu tohto modelu.
- Jednoduchosť – best-effort model nepotrebuje žiadnu špeciálnu konfiguráciu, čo z neho robí najjednoduchší model a najrýchlejší na implementáciu.

Nedostatky modelu best-effort sú nasledovné:

- Chýbajúca garancia služby – best-effort model negarantuje dostupnú šírku pásma, stratovosť alebo oneskorenie pre prenášané služby.
- Chýbajúce rozlišovanie služieb – best-effort model nerozlišuje pakety jednotlivých aplikácii, ktoré majú rôzny stupeň dôležitosti pre ich užívateľa.

2.3.2 IntServ model

Model integrovaných služieb (IntServ) bol vyvinutý v polovici deväťdesiatych rokov, kedy bol zaznamenaný prvý pokus o implementáciu QoS v počítačových sieťach požadovaný pre real-time aplikácie. IntServ je založený na explicitnom signalizovaní a manažovaní (rezervovaní) sieťových zdrojov pre jednotlivé aplikácie podľa ich potreby. IntServ je často označovaný ako "Hard-QoS", lebo garantuje charakteristiky siete ako napríklad šírku pásma, oneskorenie a stratovosť, a preto dokáže poskytovať predvídateľnú úroveň služieb.

IntServ používa pre signalizáciu protokol RSVP (Resource Reservation Protocol). Aplikácia, ktorá má špecifické požiadavky na prenosové pásmo, musí čakať na dokončenie rezervácie zdrojov pomocou protokolu RSVP pozdĺž celej cesty, ktorou bude daná komunikácia prechádzať. Pokiaľ je rezervácia úspešná, môže aplikácia pokračovať ďalej. V dobe, keď je aplikácia aktívna, smerovače na ceste, ktorou aplikácia komunikuje, poskytujú aplikácii šírku pásma, ktorú si rezervovala. Pokiaľ RSVP nedokáže úspešne rezervovať šírku pásma kompletne celou cestou, tak aplikácia nemôže ďalej pokračovať.

Pre úspešnú implementáciu IntServ, okrem podpory protokolu RSVP, je potrebné na smerovačoch v sieti zaistiť nasledujúce funkcie:

- Riadenie prístupu – Nie každá aplikácia by mala mať možnosť rezervácie zdrojov siete a množstvo rezervovaných zdrojov by nemalo byť neobmedzené. K tomuto treba implementovať funkciu, ktorá bude dohliadať nad samotným procesom rezervácie.
- Klasifikácia – Pakety patriace aplikácii, ktorá vykonala rezerváciu, musia byť rozpoznané na tranzitných smerovačoch, aby im mohla byť poskytnutá patričná garancia pásma.
- Obmedzovanie prenosovej rýchlosti – Je dôležité monitorovať, merať a obmedzovať prenosy jednotlivých aplikácii tak, aby neprekročili množstvo zdrojov, ktoré im bolo rezervované. Pokiaľ toto množstvo prekročia, tak musí byť vykonaná patričná akcia.
- Radenie do front – Je dôležité pre sieťové zariadenia, aby boli schopné pozdržať pakety v dobe zahltenia, keď sú spracovávané iné pakety. K tomu existujú rôzne mechanizmy využívajúce rôzne typy front.
- Plánovanie – Plánovanie je spojené s použitým typom fronty. Definuje spôsob, akým sú pakety vyberané z fronty a predávané na výstupné rozhranie.

Výhody modelu IntServ sú nasledovné:

- Koncepčná jednoduchosť a integrácia s riadením prístupu.
- Diskrétny typ QoS služieb založený na konkrétnych tokoch a konkrétnych aplikáciách.
- Dynamická signalizácia prenosových parametrov.

Nedostatky modelu IntServ sú nasledovné:

- Každý aktívny tok musí byť nepretržite signalizovaný kvôli stavovej architektúre protokolu RSVP. Pri veľkom množstve tokov môže samotná réžia protokolu RSVP spotrebovať nezanedbateľné množstvo prostriedkov siete.
- Všetky tranzitné smerovače musia podporovať RSVP a ďalšie podporné funkcie. Kvôli tomuto nie je model IntServ dobre škálovateľný pre veľké siete ako napríklad Internet.

2.3.3 DiffServ model

DiffServ model je najnovší z trojice prezentovaných QoS modelov. Cieľom pri jeho vývoji bolo prekonať nedostatky jeho predchodcov. DiffServ nie je garantovaný QoS model, ale je vysoko škálovateľný. Model je štandardizovaný organizáciou IETF v RFC 2474 a RFC2475. Zatiaľ čo IntServ bol nazvaný "Hard-QoS"model, DiffServ je nazývaný "Soft-QoS"modelom, kvôli svojej neschopnosti poskytovať absolútnu garanciu prenosových parametrov.

DiffServ nevyužíva žiadny signalizačný protokol. Je založený na technike PHB (Per-Hop Behavior). PHB znamená, že každý uzol v sieti musí mať vlastnú konfiguráciu zabezpečujúcu špecifickú úroveň služieb pre konkrétne triedy dátových tokov. PHB nepotrebuje signalizáciu, lebo pakety je možné pomocou ich označenia identifikovať do niektorej z požadovaných tried. Tento model je viacej škálovateľný, lebo réžia spojená so signalizáciou a udržiavaním stavových informácii nie je potrebná. Každý prvok v sieti je schopný zvládať limitované množstvo prenosových tried. To znamená, že aj keď bude naraz aktívnych tisíc dátových tokov, budú stále zaradené do jednej z preddefinovaných tried a každému toku budú priradené prenosové parametre patričnej triedy. Počet tried a úrovne služieb danej

triedy sú závislé na možnostiach používaného hardwaru a potrieb užívateľov, respektíve aplikácií.

Výhody modelu DiffServ sú nasledovné:

- Škálovateľnosť – Nie je potrebné udržiavať stavové informácie o tokoch a ich rezerváciách.
- Výkonnosť – Obsah paketu postačuje prehliadnuť iba raz. Na základe jeho obsahu môže byť označený a následné rozhodnutia pre aplikovanie QoS môžu byť vykonané na základe tejto značky.
- Interoperabilita – Možnosť použitia sieťových prvkov rôznych výrobcov.
- Flexibilita – DiffServ model nepredpisuje presné funkcie, ktoré majú byť implementované na sieťovom prvku. Funkcie môžu byť optimalizované na konkrétny hardware s tým, že stačí aby bolo dodržané očakávané správanie definované v PHB.

Nedostatky modelu IntServ sú nasledovné:

- Nemožnosť poskytovania absolútnej garancie služby.
- Samotná konfigurácia na sieťovom prvku môže byť komplexná a môže si vyžadovať vysoké odborné znalosti.

2.4 Nástroje pre zaistenie QoS v modelu DiffServ

Všeobecne nástroje pre zaistenie QoS môže rozdeliť do nasledovných kategórii:

- Klasifikácia a značkovanie
- Manažment zahltenia
- Predchádzanie zahlteniu
- Regulácia a tvarovanie prenosovej rýchlosti
- Nástroje pre zefektívnenie prenosu

2.4.1 Klasifikácia a značkovanie

Cieľom QoS je poskytovať rozličné zaobchádzanie pre rôzne sieťové toky. Preto je potrebné definovať prenosové triedy pomocou identifikácie a zoskupovania týchto sieťových tokov. Klasifikácia je práve mechanizmus, ktorý túto funkciu zaisťuje. K identifikácii tokov používa nasledujúce kritéria:

- Parametre vrstvy 1 (L1) – Fyzické rozhranie, PVC
- Parametre vrstvy 2 (L2) – MAC adresa, 802.1Q/p class of service (CoS) bity, VLAN identifikátor, MPLS experimentálne bity (MPLS EXP), ATM cell loss priority (CLP), Frame Relay discard eligible (DE) bit
- Parametre vrstvy 3 (L3) – IP Precedence, DiffServ code point (DSCP), zdrojová/cieľová IP adresa

- Parametre vrstvy 4 (L4) – TCP alebo UDP porty
- Parametre vrstvy 7 (L7) – identifikácia na základe špecifík jednotlivých aplikácií

V minulosti bola používaná klasifikácia bez značkovania. Výsledkom bolo, že každý QoS mechanizmus na každom zariadení musel klasifikovať všetky pakety na základe patričných kritérií pred tým ako mohli byť aplikované QoS politiky. Takáto opakovaná klasifikácia, napríklad na základe L7 informácií, je veľmi neefektívna. Preto sa v dnešných sieťach pri prvej klasifikácii pakety značkujú. V ideálnom prípade by počiatočná klasifikácia s následným značkovaním mala byť vykonaná čo najbližšie k zdroju paketov. Preferované miesto v sieti pre túto úlohu je na okrajových sieťových zariadeniach ako napríklad IP telefón alebo prístupový prepínač.

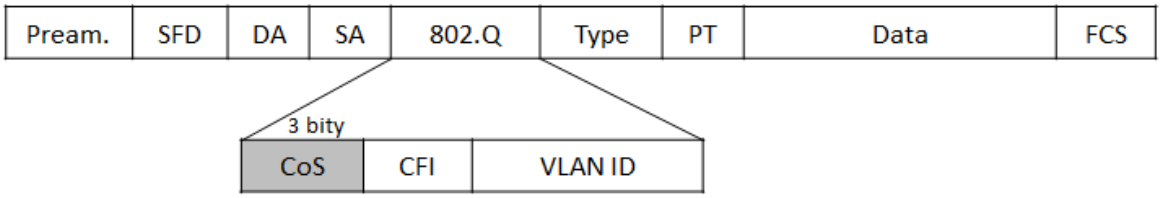
Značkovanie môže byť na úrovni L2 v hlavičke rámca, respektíve bunky alebo na úrovni L3 v hlavičke paketu. Najčastejšie používané L2 značky sú CoS (v hlavičke ISL alebo 802.1Q), EXP (v MPLS hlavičke), DE (v hlavičke Frame Relay) a CLP (v hlavičke ATM bunky). Najčastejšie používané L3 značky sú IP precendence alebo DSCP (v IP hlavičke).

CoS v 802.Q/P Ethernetovom rámci

802.1Q definované organizáciou IEEE má za úlohu umožniť prenos virtuálnych LAN sieti (VLAN) medzi sieťovými prepínačmi. 4-bajtová 802.1Q hlavička je vložená po zdrojovej MAC adrese v hlavičke ethernetového rámca. Tri bity 802.1Q hlavičky označované ako CoS (class of service) sú vyžívané na účely QoS. Môžu nadobúdať osem hodnôt zobrazených v nasledujúcej tabuľke.

CoS	**IETF RFC791**	**Aplikácia**
0	Routine	Bežné dáta
1	Priority	Stredne prioritné dáta
2	Immediate	Vysoko prioritné dáta
3	Flash	Signalizácia hovorov
4	Flash-Override	Video konferencie
5	Critical	Hlasové hovory
6	Internet	Rezervované
7	Network	Rezervované

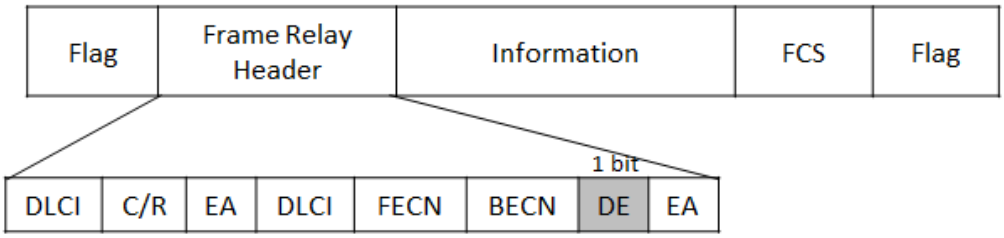
Tabuľka 2.1: Definované hodnoty CoS



Obr. 2.1: Hlavička ethernetového rámca a pozícia CoS

DE a CLP vo Frame Realay a ATM

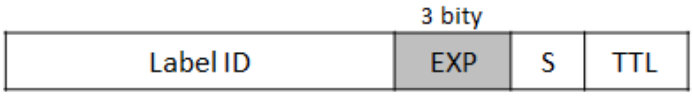
QoS štandardy pre Frame Realay a ATM boli definované a používané skôr ako organizácia IETF špecifikovala štandardy QoS používané v IP sieťach. Frame Relay rámce a ani ATM bunky nemajú v hlavičke položku porovnateľnú s trojbitovou položkou CoS, ktorú obsahuje hlavička 802.1Q. Frame Relay má jednobitovú položku DE (discard eligible) a ATM má jednobitovú položku CLP (cell loss priority), ktorá informuje prepínače či daný rámec/bunka je alebo nie je kandidátom na zahodenie v čase zahltenia.



Obr. 2.2: Hlavička Frame Relay rámca a pozícia DE

EXP v MPLS hlavičke

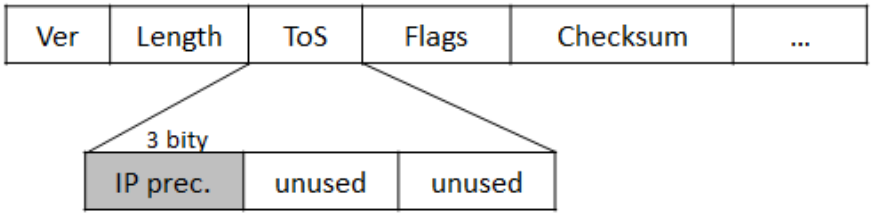
MPLS pakety sú IP pakety, ktoré majú pred IP hlavičkou vloženú jednu alebo viac MPLS hlavičiek. Položka EXP (experimental) z MPLS hlavičky je využívaná pre potreby QoS značkovania. Položka EXP je trojbitová a navrhnutá tak, aby bola kompatibilná so značkami CoS respektíve IP precendence.



Obr. 2.3: MPLS label a pozícia EXP

IP precedence a DSCP v IP hlavičke

Počiatočná špecifikácia QoS pre IP protokol publikovaná v RFC 791 definovala prvé 3 bity ToS bajtu IP hlavičky ako IP precedence. IP precedence môže nadobúdať osem rôznych hodnôt. Čím väčšia hodnota IP precedence, tým je paket viacej dôležitý a mal by byť uprednostnený pri odosielaní. Obrázok 2.4 zobrazuje IP paket so zameraním na ToS bajt, presnejšie na IP precedence bity. Osem rôznych kombinácií IP precedence s ich názvami podľa RFC 791 sú uvedené v tabuľke 2.2. IP precedence s hodnotami 6 a 7, nazývané Internetwork Control a Network Control, sú rezervované pre kontrolné sieťové protokoly a tieto hodnoty by nemali byť používané užívateľskými aplikáciami. Pre užívateľské aplikácie je dostupných zvyšných 6 hodnôt.



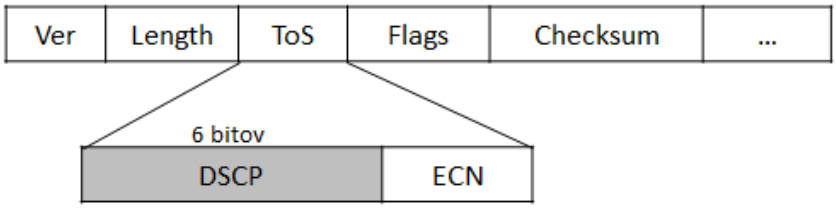
Obr. 2.4: IP hlavička a pozícia IP precedence

IP precedence (desiatkovo)	**IP precedence (binárne)**	**RFC 791 pomenovanie**
0	000	Routine
1	001	Priority
2	010	Immediate
3	011	Flash
4	100	Flash-Override
5	101	Critical
6	110	Internetwork Control
7	111	Network Control

Tabuľka 2.2: Definované hodnoty IP precedence

Predefinovanie ToS bajtu pre potreby difencovaných služieb s použitím prvých 6 bitov sa nazýva DSCP (Differentiated Services Code Point). DSCP poskytuje väčšiu flexibilitu a kapacitu pre rozlíšenie nových typov služieb a aplikácií. Zvyšné dva bity ToS bytu sa používajú pre kontrolu toku a nesú označenie ECN (Explicit Congestion Notification). DSCP je spätne kompatibilné s IP precedence, čo dáva možnosť postupného prechodu na QoS model využívajúci DSCP značkovanie. Definícia hodnôt DSCP obsahuje 4 PHB:

- Class selector PHB – S poslednými troma bitmi DSCP nastavenými na 000 zaručuje class selector PHB spätnú kompatibilitu s IP precedence. Keď DSCP kompatibilné sieťové zariadenie prijme IP paket od DSCP nekompatibilného zariadenia, kompatibilné zariadenie môže byť nakonfigurované spracovávať iba IP precedence bity. Keď IP pakety sú zasielané z DSCP kompatibilného zariadenia na zariadenie, ktoré nie je DSCP kompatibilné, tak prvé 3 bity DSCP sú nastavené ekvivalentne k IP precedence hodnote a zvyšné 3 bity sú nastavené na 0.
- Default PHB – Prvé 3 bity DSCP sú nastavené na 000 a používa sa pre best-effort služby. Pokiaľ nejaká DSCP hodnota nie je namapovaná na PHB, tak je priradená práve do default PHB.
- Assured forwarding (AF) PHB – Prvé 3 bity DSCP nastavené na 001, 010, 011 alebo 100 (sú nazývané AF1, AF2, AF3 a AF4). AF PHP sa používa pre garanciu šírky pásma jednotlivým službám.
- Expedited forwarding (EF) PHB – Prvé tri bity DSCP nastavené na 101 (kompetné DSCP je nastavené na 101110, desiatkove 46). EF PHB poskytuje daným službám nízku latenciu.



Obr. 2.5: IP hlavička a pozícia DSCP

Typ PHB	**DSCP bity**					
Class Selector PHB	-	-	-	0	0	0
Default PHB	0	0	0	-	-	0
Asured Forarding (AF) PHB	0	0	1	-	-	0
	0	0	1	-	-	0
	0	1	0	-	-	0
	0	1	1	-	-	0
Expedited Forwarding (EF) PHB	1	0	1	1	1	0

Tabuľka 2.3: Definované hodnoty DSCP

2.4.2 Manažment zahltenia

Zahltenie vzniká, keď rýchlosť prichádzajúcich paketov na rozhranie (pakety prichádzajúce v rámci vnútorného prepínania v danom zariadení) presahuje rýchlosť, ktorou je možné pakety z daného rozhrania odosielať (z rozhrania na médium). K zahlteniu dochádza napríklad v situácii, keď dátové toky do zariadenia vstupujú po rýchlejšej komunikačnej linke ako zariadenie opúšťajú (speed mismatch problem). Ďalšou situáciou kedy dochádza k zahlteniu je prípad, keď dátové toky do zariadenia vstupujú viacerými komunikačnými linkami, ale zariadenie opúšťajú jednou linkou (aggregation problem).

Sieťové zariadenie môže reagovať na zahltenie rôznymi spôsobmi, z ktorých sú niektoré jednoduché a iné zase veľmi sofistikované. Riešiť trvalé zahltenie je často možné iba navýšením šírky pásma danej linky. Na druhej strane dočasné zahltenie je efektívnejšie riešiť technikou vytvárania front ako navyšovaním šírky pásma, ktoré je typicky cenovo nákladné. Pri vytváraní front prichádzajúce pakety, ktoré z dôvodu nedostatku kapacity nemôžu byť odoslané hneď po príchode, sú dočasne pozdržané vo fronte a následne sú odoslané, keď sa dostanú na rad. Poradie, v ktorom sú pakety vyberané z front, záleží na algoritme plánovania implementovanom nad vytvorenými frontami. Pokiaľ je fronta plná, tak nové prichádzajúce pakety sú zahodené (tail drop).

Mechanizmus vytvárania front na každom fyzickom rozhraní je zložený z hardwarovej a softwarovej časti. Pokiaľ hardwarová fronta, tiež nazývaná prenosová fronta (transmit queue), nie je zahltená, tak pakety nevchádzajú do softwarovej fronty, ale sú vložené priamo do hardwarovej fronty. Odtiaľ sú pakety v čo najkratšom čase zasielané priamo na médium (na základe FIFO plánovania). Až v prípade, že je hardwarová fronta plná, tak sú pakety uložené do softwarovej fronty, odkiaľ sa následne presúvajú do hardwarovej fronty na základe zvoleného plánovania. Existuje pomerne široké množstvo plánovacích algoritmov, z ktorých tie najbežnejšie sú FIFO, PQ, CQ a WFQ predstavené následne v tejto sekcii.

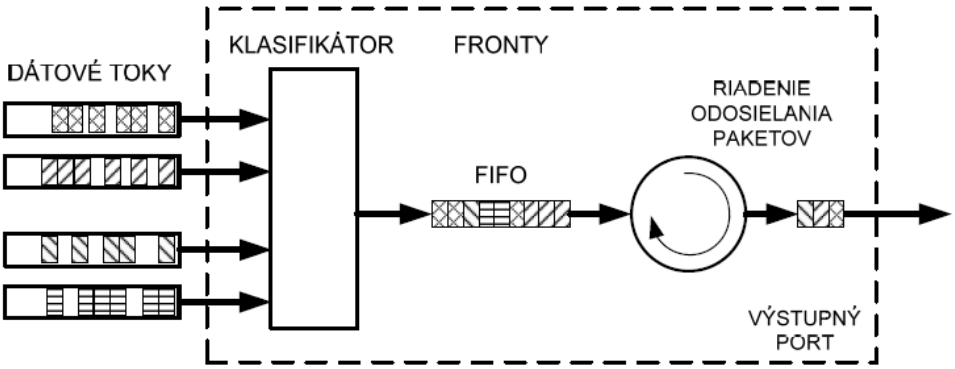
Riešenie pomocou oboch - softwarovej a hardwarovej fronty má svoje výhody. Bez softwarovej fronty by všetky pakety museli byť spracované FIFO plánovaním v hardwarovej fronte. Čo by malo za následok zamedzenie preferovanému nakladaniu s rôznymi triedami paketov. Preto napríklad prenosy real-time aplikácií by mohli byť značne znehodnotené. Podobný následok by malo aj prílišné zväčšenie veľkosti hardwarovej fronty. Naopak úplné zmenšenie veľkosti hardwarovej fronty ponechá celé plánovanie na softwarovej fronte, ale aj tento prípad má svoje nedostatky. Napríklad keď hardwarová fronta má veľkosť na udržanie len jedného paketu a aktuálny paket je z nej odoslaný na médium, je potrebné vyvolať prerušenie CPU k tomu, aby mohol byť prenesený ďalší zo softwarovej fronty do hardwarovej. V dobe, keď dochádza k tomuto prenosu, kde výber paketu môže byť na základe komplikovaného algoritmu, z hardwarovej fronty nie sú na médium odosielané žiadne dáta, čo môžeme označiť ako mrhanie šírkou pásma. A taktiež výber len jedného paketu počas jedného prerušenia zvyšuje zaťaženie CPU.

FIFO Queueing

Jedným z dôvodov prečo sieťové zariadenia potrebujú softwarové fronty je uloženie prenášaného paketu do doby, kým je paket vyslaný cez výstupné rozhranie. Zatiaľ, čo niektoré typy front vykonávajú aj iné funkcie ako preusporiadanie pomocou plánovania, FIFO (First In First Out) fronta poskytuje len možnosť uloženia paketu počas čakania na výstupné rozhranie. Stratégia výberu paketu metódou FIFO je jednoduchá. Vždy je vybraný paket, ktorý prišiel ako prvý.

FIFO fronta má úplne jednoduchú klasifikáciu a plánovanie. FIFO fronta používa iba jednu softwarovú frontu pre každé rozhranie. Keďže existuje len jedna fronta, tak nie je potrebné klasifikovať paket a rozhodnúť, do ktorej z front má byť vložený. Tiež nie je potrebná plánovacia logika nad frontami k určeniu, z ktorej fronty má byť v danom okamihu vybraný paket. Jediný zaujímavý parameter FIFO fronty je jej dĺžka. Tento parameter je konfigurovateľný a ovplyvňuje oneskorenie a stratovosť paketov.

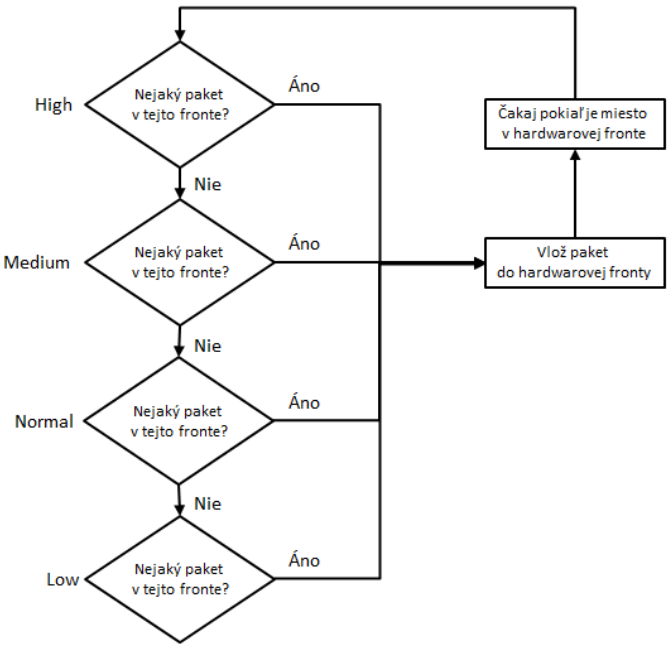
FIFO fronta používa tail drop pri rozhodnutí či paket bude zahodený alebo vložený do fronty. Pokiaľ je fronta dlhšia, tak je možné do nej vložiť viac paketov, a teda je tu menšia pravdepodobnosť, že fronta sa zaplní. Pokiaľ je menšia pravdepodobnosť zaplnenia, tak je v konečnom dôsledku nižšia stratovosť. Ale dlhšia fronta môže zvýšiť oneskorenie a rozptyl oneskorenia. S kratšou frontou je oneskorenie nižšie, ale fronta sa naplní skôr a teda dochádza k zvýšeniu stratovosti paketov. Tento fakt platí všeobecne pre všetky typy front, vrátane FIFO fronty.



Obr. 2.6: Štruktúra FIFO fronty

Priority Queueing

Hlavným rozlišovacím znakom PQ (Priority Queueing) je samotný plánovač. PQ plánuje výber paket tak, že fronta s vyššou prioritou je vždy obslúžená pred frontou s nižšou prioritou. Vedľajším efektom tohto prístupu je možnosť vyhladovania front s nižšou prioritou. Typická implementácia PQ na hardwarových smerovačoch obsahuje štyri fronty nazývané High, Medium, Normal, Low. Logika plánovača PQ môže byť jednoducho reprezentovaná nasledovným diagramom.

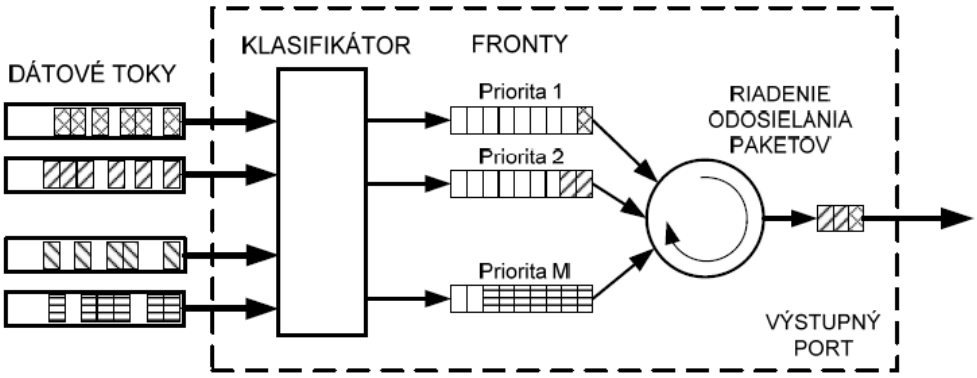


Obr. 2.7: Princíp PQ plánovania

Ako je možné z obrázku 2.7 vidieť, pokiaľ fronta High obsahuje čakajúci paket, tak plánovač stále zoberie paket práve z High fronty. Pokiaľ fronta High neobsahuje žiadny

čakajúci paket, ale fronta Medium áno, tak paket je vybraný z Medium fronty. Následne proces zase začína od High fronty. Low fronta je obslúžená len v prípade, že High, Medium a Normal fronty nemajú žiadne čakajúce pakety.

PQ plánovač má svoje výhody a nevýhody. Pakety z High fronty môžu využívať 100 percent šírky pásma s minimálnym oneskorením a rozptylom oneskorenia. Fronty s nižšou prioritou ale môžu trpieť. Prakticky to znamená, že pakety vo frontách s nižšou prioritou sú počas veľkého zahltenia linky obslúžené omnoho neskôr, ako je tomu v dobe nízkeho zahltenia. Keď sú linky zahltené, užívateľské aplikácie, ktorých pakety sú vložené do front s nižšou prioritou, môžu prestať korektne fungovať. Fakt, že v PQ dochádza k hladovaniu front s nízkou prioritou, robí tento mechanizmus málo populárnym a v reálnych sieťach prakticky nepoužívaným.

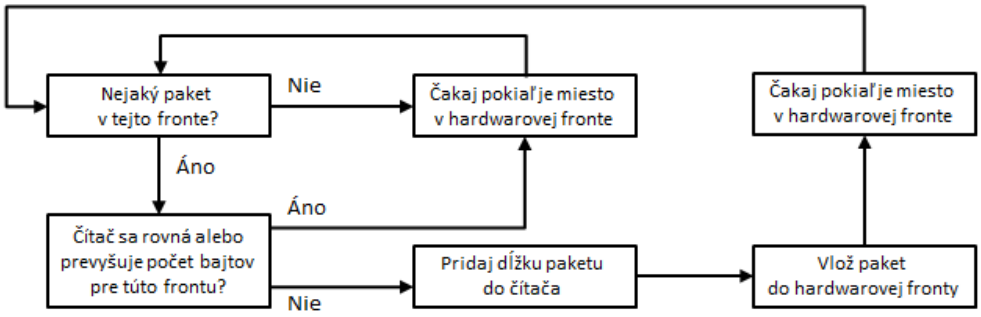


Obr. 2.8: Štruktúra PQ fronty

Custom Queueing

CQ (Custom Queueing) odstraňuje nevýhody PQ v podobe poskytnutia prostriedkov na rovnomerné obsluhovanie jednotlivých front v čase zahltenia. Typická implementácia CQ na hardwarových smerovačoch obsahuje 16 front a teda 16 kategórií, do ktorých je možné namapovať rôzne typy aplikácií. Negatívna stránka CQ oproti PQ je absencia možnosti uprednostniť pakety niektorej z front (ako napríklad High fronta v PQ). To znamená, že CQ neumožňuje poskytovanie dobrých prenosových služieb pre prenosy citlivé na oneskorenie a rozptyl oneskorenia.

Ako u väčšiny nástrojov manažmentu zahltenia, najdôležitejšou časťou je plánovač. CQ plánovač rezervuje približné percento šírky pásma pre každú z front. Nejde o presné percento kvôli tomu, že princíp funkcie plánovača je príliš jednoduchý. Obrázok 2.9 zobrazuje logiku CQ plánovača.



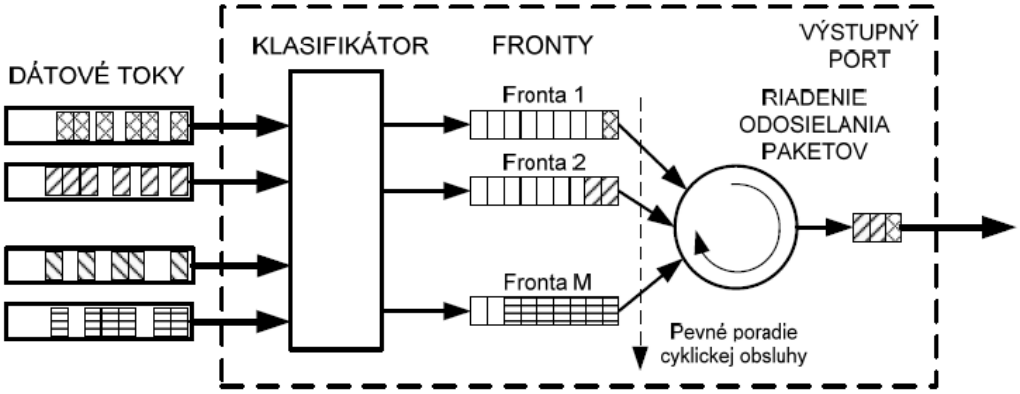
Obr. 2.9: Princíp CQ plánovania

CQ plánovač vykonáva cyklickú (round-robin) obsluhu nad každom z front, začínajúc na fonte s ID 1. CQ vyberá pakety z fronty dovtedy, kým súčet veľkostí vybraných paketov sa nevyrovná alebo nepresiahne špecifikovanú hodnotu. Po tom, čo je z fronty vybrané požadované množstvo paketov alebo vo fronte už žiadne pakety nie sú, tak plánovač prechádza na obsluhu ďalšej fronty v poradí. Tento proces sa neustále opakuje.

CQ neobsahuje nastavenie percent šírky pásma, namiesto percent obsahuje počet bajtov, ktoré majú byť z fronty odobraté počas jedného prechodu obsluhy. Napríklad môže byť použitých prvých 5 front, pričom pre každú frontu bude špecifikovaných 10000 bytov. To znamená, že pre každú frontu je teoreticky pridelených 20 percent šírky pásma. V praxi ale to nemusí zodpovedať presne 20 percetám, lebo pakety môžu mať rôznu veľkosť a môže dôjsť k situácii, že bude v jednom prechode nad frontou odobraných značne viac ako 10000 bajtov. Napríklad, keď fronta obsahuje 1500 bajtové pakety, tak podľa logiky plánovača bude z fronty odobratých 10500 bajtov.

Počet bajtov u jednotlivých front nemusí byť rovnaký. Napríklad prvé dve fronty môžu mať pridelených 5000 bajtov, ďalšie dve 10000 bajtov a piata fronta 20000 bajtov. Potom fronty 1 a 2 dostanú po 10 percent šírky pásma, fronty 3 a 4 po 20 percent a fronta 5 dostane 40 percent šírky pásma.

CQ plánovač garantuje minimálnu šírku pásma, ale umožňuje frontám za určitých okolností získať šírku pásma nad rámec garancie. Napríklad v predchádzajúcom príklade s piatimi frontami s prideleným počtom bajtov 5000, 5000, 10000, 10000 a 20000 pre fronty 1 až 5. Pokiaľ všetky fronty obsahujú značné množstvo paketov na odoslanie, tak rozdelenie šírky pásma bude 10 percent, 10 percent, 20 percent, 20 percent a 40 percent, ako to už bolo prezentované. Ale v prípade, keď budeme predpokladať, že fronta 4 neobsahuje žiadne pakety na odoslanie v danom časovom okamihu. Tak potom plánovač túto frontu automaticky preskočí a prejde k obsluhe ďalšej fronty v poradí. V tom istom časovom okamihu fronty 1 až 3 a 5 majú pakety na odoslanie, a preto dôjde k prerozdeleniu šírky pásma fronty 4 medzi tieto zvyšné fronty. V tomto prípade bude rozdelenie šírky pásma 12,5 percent, 12,5 percent, 25 percent, 0 percent a 50 percent. Z celkové počtu 16 front je v tomto príklade využitých 5 a zvyšné bude plánovač automaticky preskakovať.



Obr. 2.10: Štruktúra CQ fronty

Weighted Fair Queuing

WFQ (Weighted Fair Queuing) sa od PQ a CQ odlišuje v niekoľkých dôležitých aspektoch. Prvým asi najviac zreteľným je, že WFQ neumožňuje žiadne nastavenia klasifikácie. WFQ klasifikuje na základe tokov. Tok pozostáva zo všetkých paketov, ktoré majú rovnakú zdrojovú a cieľovú IP adresu, rovnaký zdrojový a cieľový port, rovnaký transportný protokol a rovnakú IP precedence. Teda neexistuje žiadna možnosť explicitného definovania vlastného spôsobu klasifikácie. Ďalšou veľkou odlišnosť medzi WFQ a PQ resp. CQ je plánovač, ktorý uprednostňuje toky menej náročné na šírku pásma s vysokou IP precedence pred tokmi náročnými na šírku pásma s nízkou IP precedence. Tiež z dôvodu, že WFQ je založený na tokoch a každý tok využíva inú frontu, tak počet front je omnoho vyšší – typicky to môže byť až 4096 front na jedno rozhranie. A aj keď WFQ používa tail drop, tak v skutočnosti ide o modifikovanú verziu tail drop mechanizmu.

WFQ klasifikácia
WFQ klasifikuje pakety do tokov. Toky sú identifikované nasledujúcou šesticou:

- Zdrojová IP adresa
- Cieľová IP adresa
- Protokol transportnej vrstvy (TCP alebo UDP)
- TCP alebo UDP zdrojový port
- TCP alebo UDP cieľový port
- IP precedence

Termín "tok"môže mať niekoľko rôznych významov. Príkladom môže byť PC sťahujúce web stránku. Užívateľovi sa zobrazí stránka, následne stránku číta 10 sekúnd a klikne na ďalšiu stránku. Druhá stránka sa stiahne, užívateľ ju opäť číta 10 sekúnd a prechádza na ďalšiu stránku. Všetky stránky a objekty prichádzajú z jedného webového servera a všetky stránky a objekty sú načítané pomocou jedného TCP spojenia medzi PC a serverom (v reáli by bolo týchto TCP spojení viac ako jedno, ale pre názornosť tohto príkladu môžeme túto skutočnosť ignorovať).

Z logického pohľadu existuje v tomto príklade len jeden tok, lebo je použité iba jedno TCP spojenie. Jedno TCP spojenie znamená vždy rovnaké zdrojové a cieľové IP adresy, porty a rovnaký transportný protokol, teda ide o jeden tok. Z perspektívy WFQ ale nemusí existovať ani jeden tok, alebo môžu byť tri, alebo dokonca ešte viac. WFQ definuje existenciu toku len tak dlho, kým sa pakety daného toku nachádzajú vo fronte. Napríklad, keď užívateľ 10 sekúnd číta stiahnutú stránku, smerovač na ceste medzi PC a serverom už nemá vo fronte žiadny paket, teda tok je z WFQ odstránený. V prípade, že pri prenose nedôjde k zahlteniu, tak do fronty nie je vložený žiadny paket, teda tok vo WFQ ani nie je vytvorený.

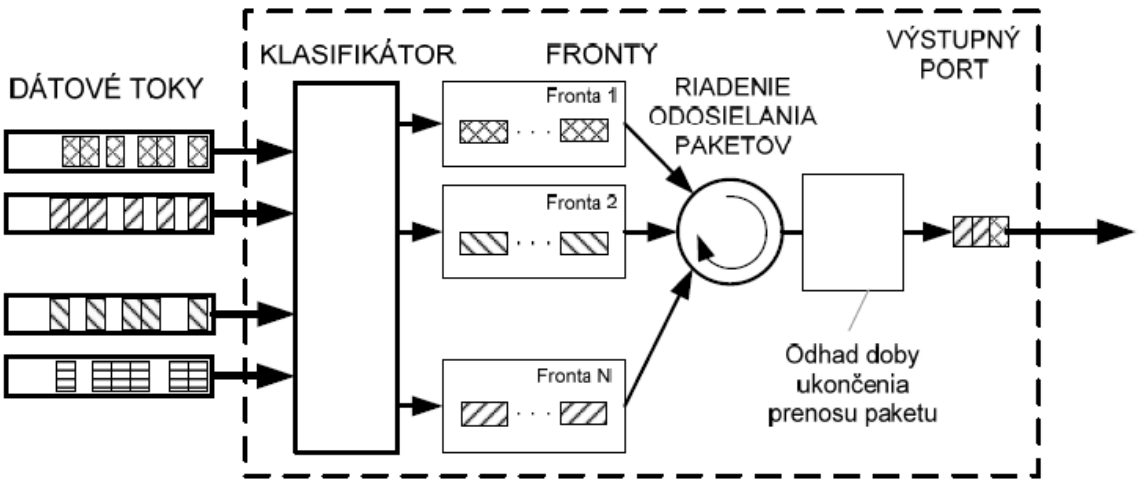
U PQ resp. CQ je daný pevný počet front, ktorý sa v čase nemení. U WFQ je počet front závislí na aktuálnom počte tokov a počet front sa dynamicky mení.

WFQ plánovač

WFQ plánovač má dva hlavné ciele. Prvým je poskytovať férové zaobchádzanie pre všetky aktuálne toky. K zabezpečeniu férovosti WFQ poskytuje každému toku rovnakú šírku pásma. Druhým cieľom je poskytovanie väčšieho množstva šírky pásma pre toky s vyššími hodnotami IP precedence.

WFQ teda každému z tokov dáva vážené percento zo šírky pásma. Rozdelenie šírky pásma sa dynamicky adaptuje počtu aktuálnych tokov. WFQ teda nemôže byť jednoducho implementované pomocou priradenia počtu bytov pre každý z tokov.

Samotný výber paketu z WFQ je úplne jednoduchý. Keď v hardwarovej fronte uvoľní pozícia, WFQ môže vybrať paket, ktorý sa na voľnú pozíciu presunie. WFQ plánovač vyberie paket s najnižším sekvenčným číslom (SN) spomedzi paketov vo všetkých frontách. SN je paketu priradené v okamihu, keď vstupuje do fronty a práve to je miesto, kde sa ukrýva celá logika WFQ plánovania.



Obr. 2.11: Štruktúra WFQ fronty

WFQ vypočítava SN každému paketu pred tým, ako je vložený do priradenej fronty. V skutočnosti WFQ vypočítava SN pred rozhodnutím o zahodení paketu, keďže SN je súčasťou modifikovanej logiky tail drop mechanizmu. WFQ plánovač zohľadňuje pri výpočte SN daného paketu jeho IP precedence a jeho dĺžku. Vzorec pre výpočet SN vkladaného paketu je nasledovný:

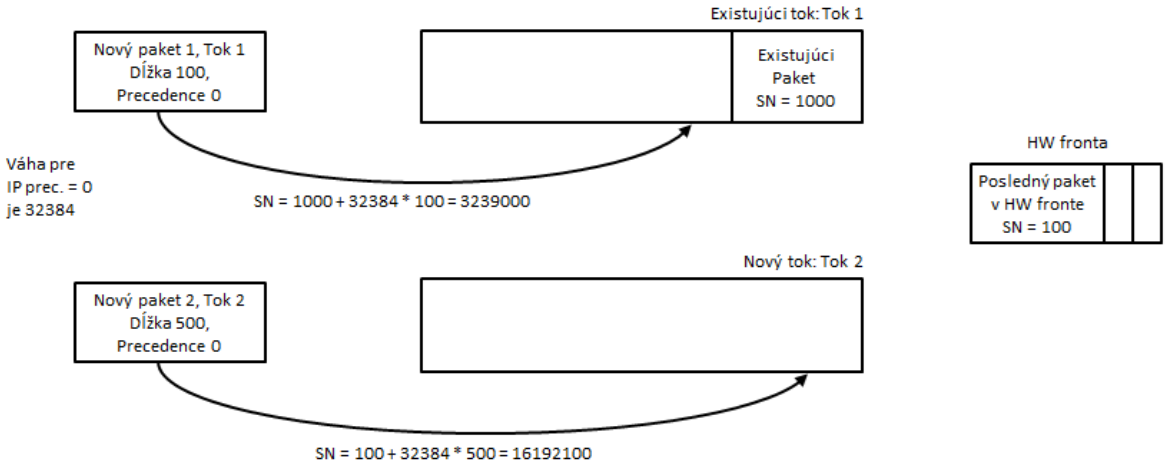
$$SN = PreviousSN + (Weight * PacketLength) \tag{2.1}$$

Váha je vypočítaná ako:

$$Weight = \frac{32384}{Precedence + 1} \tag{2.2}$$

Formula výpočtu zahrňuje hodnoty dĺžky paketu, váhy toku a predchádzajúceho SN. Zohľadnením dĺžky paketu výpočet SN rezultuje väčším číslom pre väčšie pakety a menším číslom pre menšie pakety. Zohľadnením SN naposledy pridaného paketu do danej fronty sa zaistí, že novo pridávaný paket bude mať vždy vyššie SN ako pakety, ktoré sa vo fronte už nachádzajú. Tretím komponentom formule – váhou, je zaistené pridelenie väčšej šírky pásma pre toky s vyššou IP precedence. Váha je nepriamo úmerná hodnote IP precedence, teda výsledné SN pre pakety s vyššou IP precenedce je nižšie a WFQ plánovač ich vyberie skôr a aj častejšie.

Obrázok 2.12 ilustruje vkladanie paketov do fronty. Jeden v rámci existujúceho toku a jeden nový tok.



Obr. 2.12: Princíp vkladania paketov do WFQ

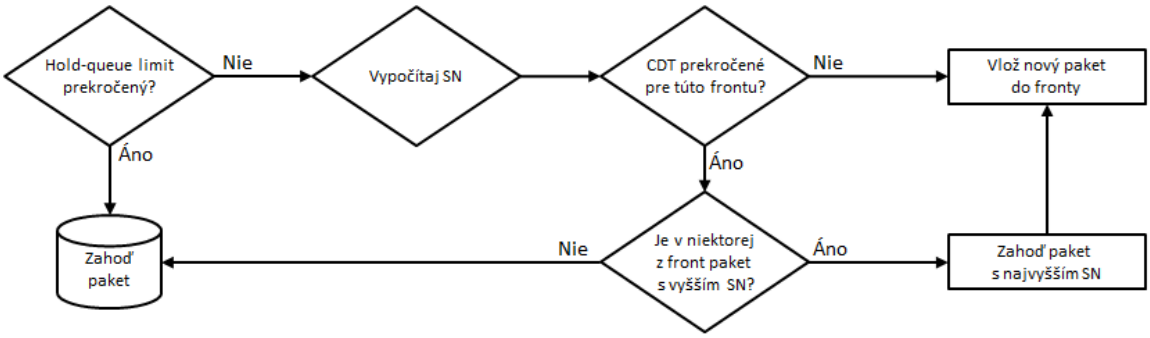
Keď je prídavaný paket 1 do toku 2, ktorý už existuje, WFQ dosadí do definovaného vzorca dĺžku pridávaného paketu (100 bytov), váhu toku (32384 pre IP precedence 0) a SN posledne pridaného paketu vo fronte daného toku (1000). Pre nový tok je použitý rovnaký vzorec. Ale vo fronte sa zatiaľ nenachádza žiadny paket, teda ako predchádzajúce SN sa musí zobrať SN naposledy vloženého paketu do hardwarovej fronty (v tomto príklade SN = 100). V oboch prípadoch WFQ priradí väčšie SN paketom väčšej dĺžky s menšou hodnotou IP precedence.

Pri veľkom množstve tokov môže WFQ trpieť problémom nazývaným "prílišná férovosť". Typicky ide o problém na linkách s menšou šírkou pásma. Keď je tokov veľa a šírky pásma je málo, WFQ sa snaží aj tak túto šírku pásma rovnomerne rozdeliť medzi všetky toky. Výsledkom môže byť situácia, kde ani jeden z tokov nemá dostatočné zdroje na to, aby prenášané aplikácie fungovali korektne.

WFQ tail drop

WFQ používa mierne modifikovaný mechanizmus tail drop pre určenie paketov, ktoré budú zahodené. Rozhodnutie je založené na niekoľkých faktoroch. Jedným z nich je aj SN daného paketu. WFQ definuje absolútny limit počtu všetkých paketov vo všetkých frontách – nesie názov hold-queue limit. Keď príde nový paket a hold-queue limit bol dosiahnutý, tak paket je zahodený. Toto rozhodnutie nie je založené na fronte daného toku, ale na fronte rozhrania ako celku.

Nasleduje rozhodnutie na základe individuálnej fronty. Pokiaľ má byť paket vložený do konkrétnej fronty a hodnota CDT (Congestive Discard Threshold) pre danú frontu bola prekročená, tak paket môže byť zahodený. CDT definuje maximálnu veľkosť fronty pre jeden tok, ale mechanizmus implementovaný v WFQ umožňuje aby paket, ktorý presahuje CDT, vošiel v niektorých prípadoch do fronty. Konkrétne ide o situáciu kedy paket presahujúci CDT má menšiu hodnotu SN ako niektorý z paketov vo frontách iných tokov. Vtedy je paket z vyšším SN vyhodený z fronty (aj keď ide o paket z iného toku) a aktuálne spracovávaný paket je vložený do fronty, aj keď presahuje CDT.



Obr. 2.13: Princíp tail drop u WFQ

2.4.3 Predchádzanie zahlteniu

Ide o pokročilé mechanizmy, ktoré pracujú na princípe detekcie vzniku zahltenia ešte skôr než k samotnému zahlteniu dôjde.

Medzi mechanizmy predchádzania zahltenia patrí:

- Predčasná detekcia s náhodnou reakciou – RED
- Predčasná detekcia s váženou náhodnou reakciou – WRED
- Explicitná signalizácia zahltenia – ECN

Základným cieľom je vyvarovať sa úplnému zaplneniu výstupnej fronty a následnému zahadzovaniu všetkých paketov – tail drop. V prípade sieťovej prevádzky založenej na protokole TCP je tail drop veľmi neefektívnou technikou. Pri strate paketov sa automaticky zníži veľkosť TCP okna a teda aj rýchlosť odosielania paketov. Problémom je, že v rovnakom okamihu nastáva strata paketov vo väčšine existujúcich TCP spojení, čo následne vedie k javu nazývanému globálna TCP synchronizácia. Priemerné využitie kapacity výstupných portov sa globálnou TCP synchronizáciu radikálne znižuje.

RED

Metóda umožňuje prvkom na sieti proaktívne reagovať na možnosť zahltenia pozorovaním aktuálnej dĺžky výstupnej fronty. Pri detekcii blížiaceho sa zahltenia začne s určitou pravdepodobnosťou náhodne vyberať pakety, ktoré budú zahodené. Tým, že zahodené pakety sú vybraté náhodne ešte pred aplikovaním tail drop, dochádza k postihnutiu len niektorých TCP spojení. Ostatné TCP spojenia môžu zatiaľ posielať plnou rýchlosťou, až kým nedôjde k zahodeniu jedného z ich paketov. Takto sa predchádza zahlteniu výstupnej fronty a globálnej synchronizácii protokolu TCP. Avšak je to aplikovateľné len u TCP spojení. UDP prenosy na predčasné zahodenie paketu nijako nereagujú a preto mechanizmus RED nemusí byť vhodný pre predchádzanie zahltenia v sieťach, kde značná časť aplikácií využíva transportný protokol UDP.

WRED

Metóda rozširuje základný mechanizmus RED o možnosť viacerých profilov výberu zahadzovaných paketov. Paket je zaradený do profilu podľa svojej váhy, ktorá sa určí zo značky paketu. Typicky sa k tomuto účelu vyžíva hodnota DSCP alebo IP precedence. Každý profil má definovanú prahovú hodnotu dĺžky fronty, kedy sa spúšťa mechanizmus RED a pravdepodobnosť s akou budú pakety vyberané. Takýmto profilovaním je možné ochrániť pakety s vysokou prioritou a najskôr aplikovať mechanizmus na pakety, ktoré nie sú až tak kritické.

ECN

ECN je metóda predchádzania zahlteniu aplikovaná na TCP prevádzku. Táto metóda využíva podstatne odlišný prístup ako metódy RED a WRED.

Hlavným rozdielom medzi RED a ECN je v tom, že v RED sú pakety náhodne zahadzované, zatiaľ čo v ECN sa náhodne vybrané pakety posielajú ďalej (nie sú zahodené) so značkou indikujúcou zahltenie koncovým staniciam. Koncové stanice by mali na túto indikáciu zareagovať redukciou rýchlosti odosielania dát. Pre značkovanie sa využívajú posledné dva bity TOS bytu. Ten istý algoritmus, ktorý používa RED, je využívaný aj pre výber paketov indikujúcich zahltenie.

2.4.4 Regulácia a tvarovanie prenosovej rýchlosti

Regulácia (policing) a tvarovanie (shaping) sú dva rôzne mechanizmy používané pre obmedzovanie množstva dát tečúcich prenosovými linkami. Oba mechanizmy merajú počet prenášaných bitov v danom intervale a porovnávajú ho s hodnotou nakonfigurovanej politiky resp. SLA (service level agreement). SLA je dohoda typicky medzi zákazníkom a poskytovateľom služieb a stanovuje parametre poskytovaných služieb, ako napríklad prenosovú rýchlosť, spoľahlivosť, dostupnosť a záležitosti týkajúce sa účtovania.

Pri tvarovaní sú pakety, ktoré presahujú stanovenú politiku, ukladané do fronty a sú následne odoslané, keď to mechanizmus tvarovania umožní. Pri regulácii sú pakety presahujúce stanovenú politiku buď zahodené, alebo sú preznačkované na nižšiu úroveň zaobchádzania. Z dôvodu, že pri tvarovaní sú pakety ukladané do fronty, tvarovanie môže byť aplikované len na pakety opúšťajúce rozhranie zariadenia. Regulácia môže byť aplikovaná v oboch smeroch ako na prichádzajúce, tak aj na odchádzajúce pakety.

Pretože pri regulácii sú pakety zahodené alebo preznačkované, tak nedochádza k zvýšeniu oneskorenia paketov presahujúcich stanovenú politiku. Pakety nad rámec politiky, ktoré

sú zahodené, musia byť znovu prenesené. Preznačkované pakety sú okamžite bez akéhokoľvek pozdržania odoslané. Oproti tomu pri tvarovaní sú pakety nad rámec politiky vložené do fronty, kde naberajú ďalšie oneskorenie v závislosti na definovanej politike a plánovacom algoritme fronty. Hlavné prínosy regulovania prenosovej rýchlosti sú nasledovné:

- Limitovanie prenosovej rýchlosti na nižšiu ako je šírka pásma na fyzickej vrstve – Je to typické, keď zákazník platí za nižšiu prenosovú rýchlosť ako je skutočná prenosová rýchlosť linky, ktorou je pripojený. V takom prípade poskytovateľ služieb aplikuje reguláciu prenosovej rýchlosti na rozhraní, cez ktoré je zákazník pripojený.

- Limitované prenosovej rýchlosti pre každú z prenášaných tried – K tomuto využitiu dochádza v prípade, že poskytovateľ služieb garantuje odlišné SLA pre rôzne triedy dátových tokov. Poskytovateľ služieb reguláciu vynucuje, aby jednotlivé triedy nepresiahli dohodnutú prenosovú rýchlosť.

- Preznačkovanie paketov – Pakety pri presiahnutí stanovenej politiky nemusia byť zahodené, ale len preznačkované a sú teda potom prenášané sieťou s nižšou garanciou kvality služieb. Preznačkovanie je možné na úrovni L2 (CoS, EXP, DE,..) a tak isto na úrovni L3 (DSCP, IP precedence).

Hlavné prínosy tvarovania prenosovej rýchlosti sú nasledovné:

- Dodržanie prenosovej rýchlosti kontraktu s poskytovateľom služieb – Aby nedochádzalo k zahadzovaniu paketov na strane poskytovateľa, zákazník aplikuje tvarovanie na všetky odchádzajúce pakety do WAN resp. Metro Ethernet siete poskytovateľa.

- Predchádzanie zahlteniu niektorej z pobočiek prepojených WAN sieťou – Typická situácia je, keď pobočky sú pripojené rôznymi prenosovými rýchlosťami a dochádza k zahlteniu liniek s nižšou prenosovou rýchlosťou. Túto asymetriu je možné riešiť aplikovaním tvarovania na výstupnom rozhraní rýchlejšej pobočky pri komunikácii s pomalšou pobočkou.

2.4.5 Nástroje pre zefektívnenie prenosu

Hlavné nástroje pre zefektívnenie prenosu sú založené na kompresii a fragmentácii. Existuje niekoľko druhov kompresie, z ktorých sa najčastejšie používa kompresia dát na úrovni L2 rámcov a kompresia hlavičiek transportných protokolov. Fragmentácia je najčastejšie používaná spolu s prekladaním (interleaving).

Kompresia dát na úrovni L2 rámcov

Ako už názov napovedá, ide o kompresiu užitočných dát zapuzdrených v prenášanom rámci. V IP sieťach teda dochádza ku kompresii celého IP paketu. Kompresia je vykonávaná na každej linke samostatne a väčšinou ide o WAN linky nad protokolmi PPP, Frame Relay alebo HDLC. Konkrétny kompresný algoritmus závisí na podpore v danom sieťovom zariadení. Medzi najčastejšie podporované patrí Stacker, Predictor a MPPC (Microsoft Point-to-Point Compression). Primárne sa od seba odlišujú rôznou záťažou CPU a náročnosťou na pamäť.

Úlohou kompresie je redukcia celkovej veľkosti prenášaného rámca, čím dôjde k zvýšeniu priepustnosti, a tiež k zníženiu oneskorenia serializácie. Zlepšenie týchto parametrov je závislé na efektivite použitého kompresného algoritmu.

Na druhej strane samotný proces kompresie a dekompresie spôsobuje oneskorenie, ktoré je závislé na komplexnosti algoritmu a na tom, či je kompresia realizovaná v softwaru alebo hardwaru. Ale v konečnom dôsledku je celkové oneskorenie redukované. K najvýraznejšej redukcii dochádza na linkách s malou šírkou pásma.

Kompresia hlavičiek

Podobné ako u predchádzajúcej kompresie, dochádza k redukcii oneskorenia serializácie a zvýšeniu priepustnosti. Rozdielom je, že nedochádza ku kompresii samotných dát, ale len hlavičiek, v ktorých sú dáta zapuzdrené; napr. TCP hlavička, RTP hlavička, UDP hlavička, IP hlavička. Kompresia hlavičiek je užitočná vtedy, keď pakety obsahujú malé množstvo užitočných dát a pomer medzi veľkosťou hlavičiek a veľkosťou dát je nezanedbateľný.

V princípe kompresia hlavičiek funguje tak, že hodnoty v hlavičkách, ktoré sa počas prenosu nemenia, sú nahradené krátkym indexom, ktorý odkazuje do tabuľky s kompletnými informáciami o hlavičkách. Po dátovej linke je prenášaný len index namiesto celej hlavičky. Na základe indexu je potom zostavená pôvodná hlavička.

Fragmentácia

Fragmentácia spôsobuje rozdelenie veľkého paketu na menšie pakety – fragmenty. Tieto fragmenty sú sieťou prenášané ako samostatné jednotky a po prijatí ich cieľová stanica poskladá do pôvodného paketu. Fragmentácia sa používa spolu s prekladaním, ktoré umožní aby fragmenty jedného paketu nemuseli byť zasielané za sebou, ale aby mohli byť medzi nimi odoslané aj iné fragmenty, respektíve pakety.

Cieľom fragmentácie a prekladania je zabezpečiť, aby sa paket s vysokou prioritou príliš nezdržal vo fronte v situácii, kedy musí čakať na veľký dátový paket, ktorý je už v čase plánovania prioritného paketu v hardwarovej fronte. Fragmentácia a prekladanie zaručia, že v dobe plánovania prioritného paketu bude v hardwarovej fronte len fragment veľkého dátového paketu a prioritný paket môže byť naplánovaný skôr, ako zvyšné fragmenty nachádzajúce sa v softwarovej fronte.

Kapitola 3

OMNeT++ a INET Framework

Kapitola popisuje architektúru simulačného nástroja OMNeT++ a spôsob vytvárania modulov. Ďalej sa venuje nadstavbe INET Framework, ktorá obsahuje sadu simulačných modulov pre rodinu protokolov TCP/IP. Popisuje jednotlivé moduly a spôsob vytvárania simulačných modelov.

3.1 Simulátor OMNeT++

OMNeT++[7] je objektovo orientovaný, modulárny, diskrétny simulátor s otvorenou architektúrou. Je využívaný najmä v simulácii telekomunikačných a počítačových sietí a k modelovaniu rôznych sieťových protokolov. Vďaka svojej flexibilnej architektúre však umožňuje simuláciu radu iných systémov, ako napr. validácia hardwarových architektúr, vyhodnocovanie výkonnosti komplexných softwarových systémov, či dokonca modelovanie rôznych obchodných procesov. Taktiež má implicitnú podporu pre paralelnú distribuovanú simuláciu (napr. MPI).

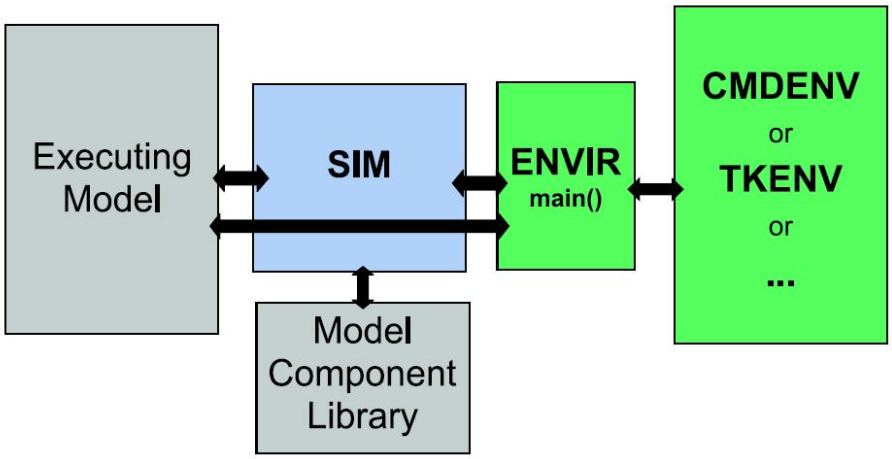
OMNeT++ poskytuje silnú podporu pre grafické rozhranie, čo mu zaručuje obľubu u užívateľov. Svedčí o tom aj viac ako 40 simulačných modelov, tzv. frameworkov, ktoré rozširujú jeho možnosti do rôznych oblastí, napr. simulácia lokálnych a internetových sietí – INET Framework, simulácia mobilných a bezdrôtových sietí – Mobility Framework či simulácia senzorových sietí – NesCT.

História OMNeT++ začala v roku 1992 na Technickej Univerzite v Budapešti. Hlavnú zásluhu na jeho vývoji má András Varga. V roku 2001 sa projekt dostal pod záštitu Univerzity v Karlsruhu. V dobe písania tejto práce je aktuálna verzia 4 a neustále je vyvíjaná.

Ako už bolo uvedené, OMNeT++ je diskrétny simulátor. Znamená to, že zmeny v simulačnom systéme sa odohrávajú v určitom okamihu spojitého alebo diskrétneho simulačného času. Každá zmena v simulačnom systéme (udalosť) je naplánovaná v kalendári udalostí a je určená časovou známkou (timestamp) a prioritou. V priebehu simulácie sa z kalendára udalostí vyberajú udalosti s najnižšou časovou známkou. Pokiaľ je časová známka rovnaká u viacerých udalostí, vyberie sa udalosť s najvyššou prioritou a tá sa vykoná. V prípade, že aj priority sa zhodujú, OMNeT++ vyberie udalosť na základe poradia v akom boli vkladané do kalendára udalostí. Doba spracovania udalosti je vzhľadom k simulačnému času nulová. Časove úseky simulačného času, v ktorých nenastáva žiadna zmena systému, sú jednoducho preskočené.

3.1.1 Architektúra OMNeT++

OMNeT++ má modulárnu architektúru. Obrázok 3.1 zobrazuje jej "high-level" pohľad. Sú v ňom naznačené jednotlivé stavebné moduly simulátora a ich vzájomná interakcia. Moduly budú podrobne popísané v nasledovnom texte.



Obr. 3.1: Architektúra OMNeT++

Sim

Modul obsahuje simulačné jadro a knižnice tried, ktoré sú využívané počas simulácie. Knižnice sa prilinkujú k samotnej simulácii staticky alebo dynamicky.

Envir

Je ďalší modul, ktorý obsahuje knižnice zdieľané všetkými užívateľskými rozhraniami. Jeho súčasťou je funkcia *main()* určujúca vstupný bod programu a obsahuje cyklus pre výber udalostí z kalendára udalostí. Envir poskytuje funkcie pre spracovanie konfiguračných súborov ".ini" jednotlivých implementácii užívateľského rozhrania. Okrem toho sa stará o zachytenie výnimiek a riešenie chýb, ktoré by mohli nastať počas simulácie v jadre. Taktiež zaisťuje načítanie potrebných modulov pri inicializácii, zápis do výstupných súborov a výpis správ pri ladení.

Cmdenv a Tkenv

Sú to moduly špecifikujúce užívateľské rozhranie. Cmdenv je užívateľské rozhranie pre spúšťanie simulácie z príkazového riadku. Vyznačuje sa svojou rýchlosťou a možnosťou spúšťania dávok simulácií pomocou skriptov. Naproti tomu Tkenv je grafické užívateľské rozhranie implementované v multiplatformovom prostredí Tcl/Tk. Jeho výhodou je interaktívne spúšťanie simulácie s možnosťou krokovania a následným vizuálným znázornením jej výsledkov.

Model Component Library

Obsahuje definície všetkých jednoduchých NED modulov simulácie a ich implentáciu v programovacom jazyku C++. Ďalej obsahuje zložené moduly, kanály, siete a správy. Zjednodu-

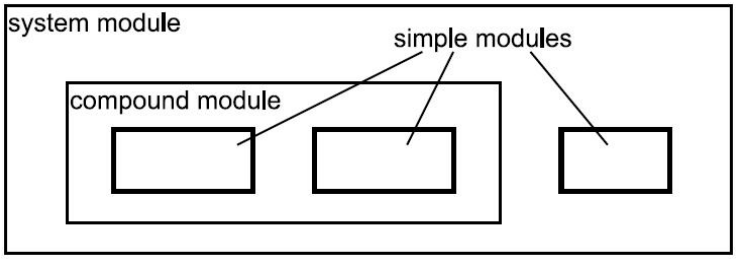
šene povedané, obsahuje všetko, čo patrí k samotnému simulovanému modelu a všetko čo bolo prilinkované k simulačnému programu.

Executing model

Reprezentuje model vytvorený pre danú simuláciu. Obsahuje objekty, ktoré sú inštanciami komponent definovaných v Model Component Library. Prístup k týmto objektom je realizovaný cez globálny objekt simulation, ten tvorí koreň stromu všetkých objektov.

3.1.2 Modelovanie v OMNeT++

Každý model simulovaný v OMNeT++ je zložený z hierarchicky usporiadaných modulov, ktoré spolu komunikujú pomocou zasielania správ. Na najvyššej úrovni je systémový modul. Systémový modul obsahuje podmoduly, ktoré tiež môžu obsahovať vlastné podmoduly. Hĺbka zanorenia modulov nie je obmedzená. Moduly obsahujúce podmoduly sa nazývajú zložené moduly (Compound Modules). Naopak moduly na najnižšej úrovni hierarchie, ktoré už neobsahujú žiadne podmoduly sa nazývajú jednoduché moduly (Simple Modules). Správanie jednoduchých modulov sa implementuje jazykom C++. Štruktúra modelu v OMNeT++ sa popisuje pomocou interného jazyka NED. Príklad takejto štruktúry je znázornený obrázku 3.2.



Obr. 3.2: Hierarchia vytváraných modulov v OMNeT++

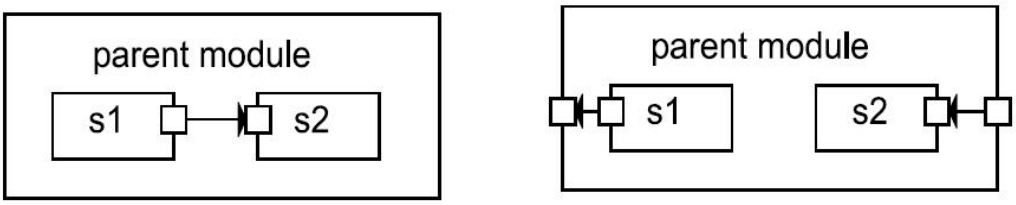
Správy, brány, linky

Moduly komunikujú výmenou správ. Správy môžu obsahovať ľubovoľné množstvo prenášaných dát. Jednoduché moduly umožňujú zasielanie správ priamo cieľovému modulu, alebo cez preddefinované cesty zložené z brán (Gates) a spojení (Connections). Brány sú vstupné a výstupné rozhrania. Správy sú posielané cez výstupné a prijímané cez vstupné brány. Spojenia, tiež nazývané linky, spájajú jednotlivé brány medzi sebou, a to vždy na úrovni jedného modulu. To znamená, že v rámci zloženého modulu je možné spojiť brány medzi jednotlivými podmodulmi alebo medzi podmodulom a rodičovským modulom (obrázok 3.3).

Prenos správ (paketov)

Každému spojeniu môžu byť definované tri parametre, ktoré umožňujú modelovanie komunikácie po sieti, ale môžu byť užitočné aj pre iné modely. Sú to nasledujúce parametre:

- Oneskorenie (propagation delay) – časový rozdiel medzi odoslaním a prijatím správy.



Obr. 3.3: Spojenia medzi bránami modulov

- Chybovosť (bit error) – pravdepodobnosť, že správa nie je správne doručená.
- Rýchlosť (data rate) – počet bitov prenesených za jednu sekundu.

Konfigurácia modelu

Jednotlivým modulom je možné priraďovať parametre, ktoré umožňujú ovplyvňovať ich správanie. Parametre môžu byť zakomponované priamo v definícii modulu (súbore NED), alebo načítavané pri štarte simulácie z konfiguračného súboru (*omnetpp.ini*).

Parametre môžu byť rôzneho dátového typu: reťazce, celé čísla , logické hodnoty, ale aj XML súbory. V rámci číselných parametrov je možné zavolať funkcie jazyka C pre generovanie pseudonáhodných čísel s rôznym rozložením, alebo interaktívne získať vstup od užívateľa.

3.2 INET Framework

INET Framework[1] je balík simulačných modelov pre rodinu protokolov TCP/IP. Konkrétne obsahuje implementáciu protokolov IPv4, IPv6, TCP, UDP a niekoľko ďalších aplikačných protokolov. Framework taktiež obsahuje model MPLS a modely linkovej vrstvy PPP, Ethernet a 802.11. Statické smerovanie je možné nastaviť pomocou modulu autokonfigurácie, alebo je možné použiť implementáciu jedného zo smerovacích protokolov. INET Framework má podporu pre bezdrôtové a mobilné simulácie. Táto podpora bola prevzatá z Mobility Frameworku.

Predchodca INET Framework bol vyvíjaný na univerzite v Karlsruhe v rokoch 2000 a 2001 pod názvom IPSuite. V roku 2003 prevzal vývoj Andras Varga, ktorý jednotlivé moduly zreorganizoval, zdokumentoval, a mnohé od základu prepísal. V roku 2004 boli pridané rozšírenia TCP implementácie a pôvodný názov sa zmenil na INET Framework. Vývoj prebieha aj v súčasnosti a hlavné zameranie patrí protokolom 802.11 a IPv6.

3.2.1 Architektúra INET Framework

INET Framework je postavený nad OMNET++ a používa rovnaký koncept – moduly komunikujúce zasielaním správ. Jednotlivé protokoly sú reprezentované jednoduchými modulmi. Rozhrania jednoduchých modulov sú popísané v NED súboroch a implementácia v súboroch s C++ kódom. Tieto jednoduché moduly môžu byť voľne kombinované a spájané do zložitých modulov pomocou jazyka NED. Veľa takýchto modulov je už vytvorených, napríklad StandardHost, Router, OSPFRouter, WirelessAP. Samozrejme každý si môže vytvoriť vlastný modul šitý na mieru pre svoju simuláciu. Hlavičky protokolov a formát paketov

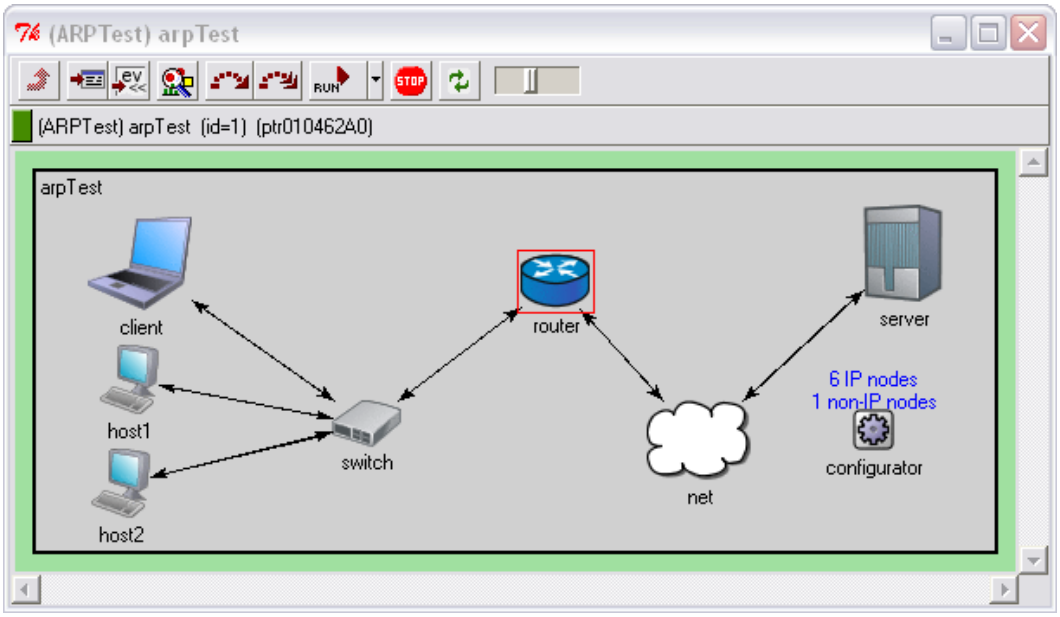
je popísaný v MSG súboroch (message definition files), ktoré sú preložené do C++ tried pomocou OMNeT++ *opp_msgc* nástroja.

Kompiláciou INET Framework vznikne binárny súbor (*bin/INET.exe* pre platformu Windows a *bin/INET* pre unixové systémy), na ktorom je založený princíp používania frameworku. Program vznikne kompiláciou všetkých jednoduchých modulov a ich prilinkovaním k simulátoru OMNeT++ (ten je pre preklad nutný). Pri spúšťaní programu sa z aktuálneho adresára načíta obsah konfiguračného súboru *omnetpp.ini*. Jedným z parametrov uložených v súbore je odkaz na hlavný NED súbor, ten následne obsahuje odkazy na ďalšie NED definície modulov, z ktorých je modul zložený.

3.2.2 Simulovanie v INET Framework

Samotné vytváranie konkrétnej simulácie má základnú ideu stále rovnakú. Už implementované moduly je potrebné ako stavebnicu poskladať do jedného celku, ktorý sa bude navonok tváriť ako modelovaný systém. Detaily sú však skoro stále odlišne. Modul od modulu má iné parametre a na ich nastavení stojí správanie celej simulácie.

V tejto časti bude na jednoduchej ukážke ARP testu demonštrovaný základný princíp vytvárania a sledovania simulácie. Uvedený príklad je súčasťou demonštračných simulácii INET Framework v zložke *Examples/Ethernet/ARPTest*.



Obr. 3.4: Topológia simulácie ARPTest

Vytváranie simulácie

Základom simulácie je súbor NED, ktorý definuje simulačný model. Určuje, ako sú jednotlivé moduly poprepájané, a môže obsahovať definíciu parametrov.
Komentovaná ukážka súboru *ARPTest.ned*:

```
//
// importovanie použitých modulov
//
```

```
import inet.networklayer.autorouting.FlatNetworkConfigurator;
import inet.nodes.ethernet.EtherSwitch;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;
import ned.DatarateChannel;
//
// vytvorenie modulu simulácie
//
network ARPTest
{
//
//definovanie vlastnosti linky
//
    types:
        channel fiberline extends DatarateChannel {
            delay = 1us;
            datarate = 512Mbps; }
//
//
// deklarácia podmodulov a ich parametrov
// sú vytvorené 4 moduly PC (StandardHost), 1 switch (EtherSwitch), 1 router (Router)
//  modul FlatNetworkConfigurator slúžiaci na konfiguráciu IP adries
//
    submodules:
        client: StandardHost {
            @display("p=71,64;i=device/laptop_l");}
        host1: StandardHost {
            @display("p=65,131;i=device/pc"); }
        host2: StandardHost {
            @display("p=60,191;i=device/pc"); }
        switch: EtherSwitch {
            @display("p=202,156"); }
        net: Router {
            @display("p=394,166"); }
        router: Router {
            @display("p=311,74"); }
        server: StandardHost {
            @display("p=512,58;i=device/server_l"); }
        configurator: FlatNetworkConfigurator {
            @display("p=495,160"); }

    connections:
//
// prepojenie jednotlivých modulov
//
        client.ethg++ <--> switch.ethg++;
        switch.ethg++ <--> host1.ethg++;
        switch.ethg++ <--> host2.ethg++;
        router.ethg++ <--> switch.ethg++;
        router.pppg++ <--> fiberline <--> net.pppg++;
        server.pppg++ <--> fiberline <--> net.pppg++;
}
```

Ďalším súborom, potrebným pre spustenie simulácie, je konfiguračný súbor. V súbore je možné priradiť parametre jednotlivým modulom (jednoduchým aj zloženým). Taktiež umožňuje nastavovať správanie prostredia simulácie. Implicitne sa súbor volá *omnetpp.ini*, ale je možné vytvoriť aj súbor s iným názvom, a potom spúšťať simuláciu s parametrom *–f <názov súboru>*.

Skrátená komentovaná ukážka súboru *omnetpp.ini*:

```
#
# obecné nastavenie pre všetky rozhrania a behy
#
[General]
  network = arpTest   # meno požitého systémového modulu
  preload-ned-files = *.ned @../../../nedfiles.lst  # dynamicky načítané moduly

  warnings = yes   # povolenie výstrah
  sim-time-limit = 500s   # maximálný simlačný čas
  cpu-time-limit= 600s   # maximálny strojový čas

#
# nastavenie grafického rozhrania Tkenv
#
[Tkenv]
  plugin-path=../../../Etc/plugins  # cesta k zásuvným modulom
  default-run=1  # implicitný beh
#
# obsahuje nastavenie parametrov pre prvý beh
# je možné definovať viacej behov, výber behu je realizovaný pomocou dialógu v Tkenv
#
[Run 1]
# parametre modulu udpApp
# ** zodpovedá žiadnemu alebo niekoľkým znakom (spôsob pridelenia parametrov viacerým
#  modulom)
  **.numUdpApps=0
  **.udpAppType="UDPBasicApp"

# parametre modulu tcp
# pre výklad jednotlivých parametrov je najlepšie nahliadnuť do dokumentácie INET
  **.tcp.mss = 1024
  **.tcp.advertisedWindow = 14336  # 14*mss
  **.tcp.sendQueueClass="TCPVirtualDataSendQueue"
  **.tcp.receiveQueueClass="TCPVirtualDataRcvQueue"
  **.tcp.tcpAlgorithmClass="TCPReno"
  **.tcp.recordStats=true
```

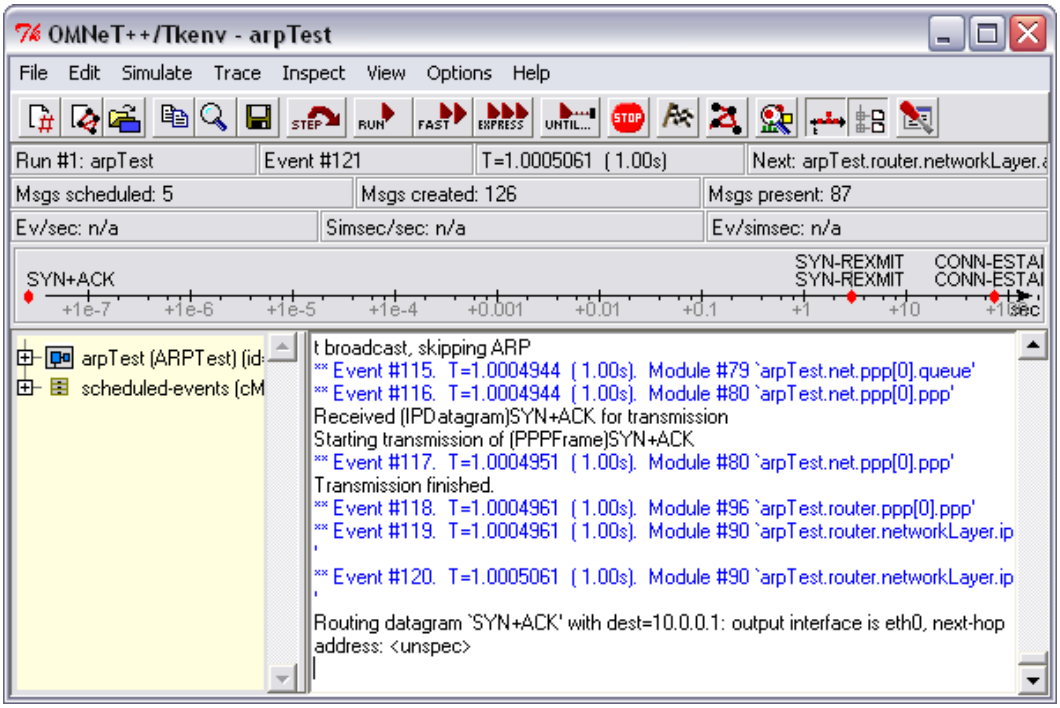
Pre spúšťanie simulácie je dobré si vytvoriť dávkový súbor BAT, ktorý zariadi korektné načítanie modulov a parametrov. Tento súbor nie je povinný.
Ukážka súboru *ARPTest.bat*:

```
..\..\..\bin\INET %*
```

Priebeh simulácie

Simuláciu je možné v grafickom prostredí spustiť v rôznych režimoch, ktoré sa líšia rýchlosťou spracovania udalostí. Na výber sú režimy “Step”, “Run”, “Fast”, “Expres” a “Run until”.

V priebehu simulácie sú do hlavného okna zapisované informácie o vykonávaných udalostiach a ich výstupy (pokiaľ sa nejaké generujú). Je možné si zobraziť kalendár udalostí a prezrieť si, ako sú jednotlivé udalosti naplánované. Pre každý modul, či už jednoduchý alebo zložený, je možné zobraziť tzv. aktívny inšpektor, ktorý pre zložený modul graficky zobrazuje jeho kompozíciu a pre jednoduchý modul zobrazuje hodnoty jednotlivých premenných alebo výstup modulu.



Obr. 3.5: Hlavné okno simulácie

Simulácia končí, keď sú spracované všetky udalosti, alebo je dosiahnutý maximálny časový limit simulácie (pokiaľ bol nastavený v súbore *omnetpp.ini*). Pre reštart simulácie slúži voľba "Rebuilt network" z menu "Simulation".

Výsledky simulácie je možné zaznamenávať do súborov SCA a VEC, z ktorých sa neskôr dajú vytvárať grafy.

Kapitola 4

Simulovanie QoS v INET Framework

Kapitola podrobne analyzuje možnosti simulácie QoS v prostredí INET Framework. V prvej sekcii je predstavený modul ANSARouter, ktorý je kľúčový z pohľadu simulovania QoS v rámci tejto práce. V druhej sekcii je potom hlbšie preskúmaný modul OutputQueue, ktorý obsahuje všetky aktuálne dostupné nástroje pre simuláciu QoS implementované v INET Framework.

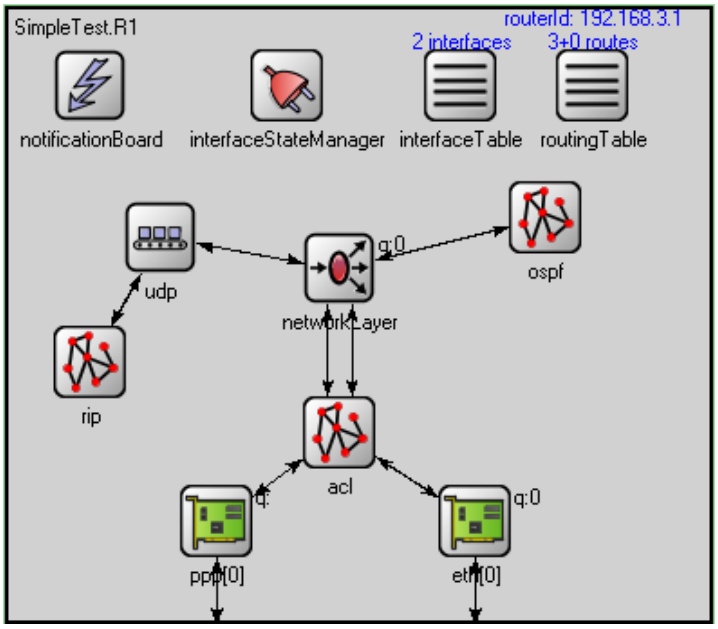
4.1 Modul ANSARouter

Táto práca je zameraná na simuláciu QoS v IP sieťach. Základom pre túto simuláciu je modul ANSARouter. Bol vytvorený v rámci projektu ANSA a hlavnou myšlienkou pri jeho vytváraní bola možnosť jednoduchého prevedenia konfigurácie z reálneho smerovača do simulačného prostredia so zachovaním čo najvyššieho stupňa hodnovernosti jeho správania. Jeho definícia sa nachádza v súbore *src/ansa/ANSARouter.ned*. Samotný modul neimplementuje žiadne správanie kódom v C++, ale iba zastrešuje ostatné moduly nižšej úrovne a vytvára funkčný celok. Vnútorná štruktúra je zobrazená na obrázku 4.1.

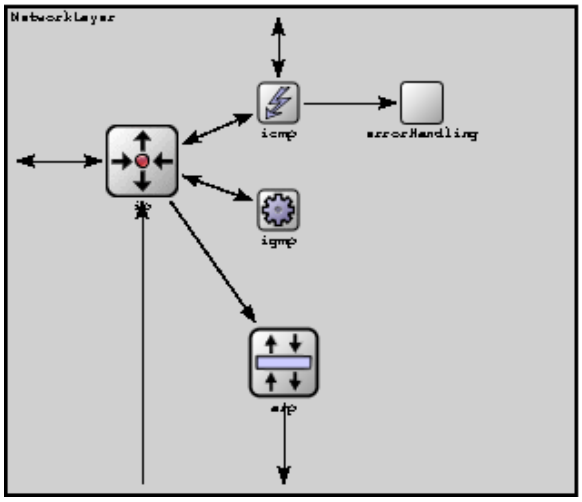
V nasledujúcej časti budú popísané jednotlivé moduly, z ktorých je modul ANSARouter zložený.

4.1.1 NetworkLayer

Ide tiež o zložený modul, ktorý reprezentuje sieťovú vrstvu. Jeho vnútorná štruktúra je zobrazená na obrázku 4.2.



Obr. 4.1: Vnútorná štruktúra modulu ANSARouter



Obr. 4.2: Vnútorná štruktúra modulu NetworkLayer

Modul sa skladá z nasledujúcich jednoduchých modulov:

Modul IP

Implementuje IP protokol. Hlavička protokolu je definovaná v IPDatagram.msg. Umožňuje komunikáciu s viacerými protokolmi vyššej vrstvy. Protokol, ktorému bude paket predaný, určuje položka "Protocol"v hlavičke paketu. Mapovanie čísla protokolu na výstupnú bránu je definované v reťazci protocolMapping. Výber rozhrania linkovej vrstvy, na ktoré má byť paket predaný, je realizovaný pomocou modulu RoutingTable.
Keď IP modul potrebuje zistiť rozhranie, jednoducho zavolá funkciu modulu RoutingTable

findBestMatchingRoute(destAddress), ktorá mu na základe cieľovej adresy vráti identifikátor rozhrania.

ARP

Implementuje ARP (Address Resolution Protocol) pre IPv4 a IEEE 802 6-bajtové MAC adresy. Používa sa pre zisťovanie MAC adresy z IP adresy na broadcastových rozhraniach, napr. Ethernetových. Či je rozhranie broadcastové, je uvedené v zázname daného rozhrania v module InterfaceTable (je možné testovať funkciou *isBroadcast()*). Pre PPP rozhrania nemá ARP zmysel.

ICMP

Implementuje protokol ICMP (Internet Control Message Protocol). Prijíma, spracuváva a odosiela odpovede správ ICMP echo a ICMP timestamp. Ignoruje správy ICMP destination unreachable, ICMP time exceeded, ICMP parameter problem a ICMP redirect.

ICMP

Je modul vyhradený pre protokol IGMP, ktorý slúži na prihlasovanie do multicastových skupín.

ErrorHandling

Spracuváva správy o chybách, ktoré prídu od ostatných protokolov. Terajšia implementácie iba vypíše oznámenie na výstup a správu zruší.

4.1.2 OSPFRouting

Implementuje smerovací protokol OSPF podľa RFC 2328. Vďaka tomuto modulu môže ANSARouter získavať informácie o dostupných sieťach od ostatných smerovačov a dynamicky si tak vytvárať smerovaciu tabuľku. Sú podporované rôzne typy sietí ako napr. Broadcast, Point-to-Point alebo Non-Broadcast. Tiež je implementovaná podpora vytvárania oblastí (area) a tzv. stub sietí.

4.1.3 RIPRouting

Podobne ako modul OSPFRouting, tak aj v prípade modulu RIPRouting ide o implementáciu smerovacieho protokolu. V tomto prípade protokolu RIP verzie 1 podľa RFC 1058.

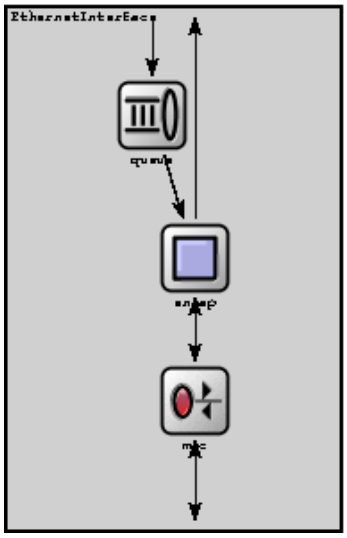
4.1.4 ACL

Modul ACL (Access Control List) umožňuje filtrovanie dátových paketov. Konkrétne umožňuje filtrovať IP pakety verzie 4. Je možné vytvárať pravidlá na základe informácií z IP hlavičky a hlavičiek transportných protokolov TCP resp. UDP. Vytvorené pravidlá je možné aplikovať na ktorékoľvek fyzické rozhranie, a to buď v smere prichádzajúcich paketov alebo odchádzajúcich paketov.

Modul je vložený medzi modul sieťovej vrstvy a moduly jednotlivých fyzických rozhraní. Všetky pakety, či už prichádzajúce alebo odchádzajúce musia prejsť týmto modulom. Implicitne, keď nie sú definované žiadne pravidlá, sú pakety prepúšťané bez filtrovania.

4.1.5 EthernetInterface

Zložený modul, ktorý predstavuje ethernetové sieťové rozhranie. Jeho vnútorná štuktúra je zobrazená na obrázku 4.3.



Obr. 4.3: Vnútorná štruktúra modulu EthernetInterface

Modul sa skladá z nasledujúcich jednoduchých modulov:

EtherMAC

Implementuje vrstvu MAC, ktorá realizuje vysielanie a príjem rámcov. Modul nerealizuje zapuzdrovanie, o túto funkciu sa starajú moduly EtherLLC alebo EtherEncap. Podporované sú rôzne verzie 10Mb Ethernet, 100Mb Ethernet half/full duplex a 1Gb Ethernet. Počas spracuvávania rámcov prijatých od vyšších vrstiev sú doplnené zdrojové adresy, pokiaľ chýbajú, následne je rámec zaradený do fronty, kde ostáva dovtedy, kým nie je prenesený. Prenos je modelovaný podľa protokolu CSMA/CD. Pokiaľ požiada vyššia vrstva, tak modul môže odoslať rámec PAUSE (používaný v prepínačoch). Príjem rámcov zo siete taktiež prebieha protokolom CSMA/CD. Modul skontroluje CRC. V prípade, že nastane chyba, tak je rámec zahodený. V promiskuitnom móde spracuváva všetky prijaté rámce, inak iba rámce so zodpovedajúcou MAC adresou alebo broadcastové rámce. Na začiatku každej simulácie dochádza medzi jednotlivými modulmi k autokonfigurácii, počas ktorej dochádza k výberu prenosovej rýchlosti a duplexu. V súčasnej implementácii nie je možné počas simulácie dynamicky pripojiť alebo odpojiť MAC rozhranie.

EtherEncap

Modul zapuzdruje pakety prichádzajúce od vyšších vrstiev do Ethernet II rámcov a posiela ich ďalej modulu MAC. Pre rámce prichádzajúce od MAC modulu je operácia opačná.

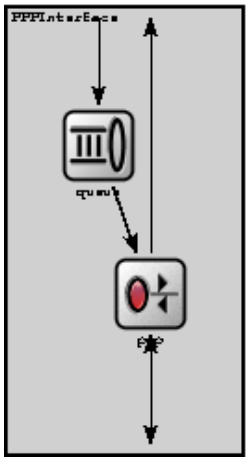
OutputQueue

Implementuje výstupnú frontu sieťového rozhrania. Je to modul, v ktorom sú implementované aktuálne dostupné nástroje pre zabezpečenie QoS. Tento modul bude podrobne

zanalyzovaný v sekcii 4.2.

4.1.6 PPPInterface

Zložený modul, funkcionálne veľmi podobný modulu EthernetInterface. V tomto prípade však ide o modelovanie dvojbodového spoja, nie zdieľaného segmentu, ako je to u EthernetInterface. Obrázok 4.4 zobrazuje vnútornú štruktúru PPPInterface.



Obr. 4.4: Vnútorná štruktúra modulu PPPInterface

Modul sa skladá z nasledujúcich jednoduchých modulov:

PPP

Implementuje protokol PPP. PPP je komplexný protokol so silnou podporou konfigurácie linky a jej údržby. Tento modul všetky tieto detaily ignoruje a jedinou jeho funkciou je zapuzdrovať rámce. Rámce po zapuzdrení posiela na výstupnú frontu. Prijímané rámce nie sú zaraďované do žiadnej fronty, ale sú okamžite spracované.

OutputQueue

Je modul totožný s modulom používaným v EthernetInterface.

4.1.7 InterfaceTable

Modul udržuje záznamy o jednotlivých sieťových rozhraniach. Rozhrania sú dynamicky registrované príslušnými L2 modulmi (napr. PPPInterface). Okrem toho sa automaticky vytvorí rozhranie loopback. Tabuľka rozhraní obsahuje iba na protokole nezávislé informácie. Všetky IPv4 alebo IPv6 špecifické informácie sú udržiavané v moduloch RoutingTable, respektíve RoutingTable6.

Tento modul nemá žiadne vstupné ani výstupné brány. Celá funkcionalita je prístupná cez členské funkcie C++ triedy.

4.1.8 RoutingTable

Modul udržiava smerovaciu tabuľku. Pre smerovače musí byť nastavený parameter routerId. Vo väčšine prípadov je ako routerId nastavená IP adresa rozhrania Loopback. Je možné zvoliť aj automatický výber najvyššej IP adresy aktívneho rozhrania.

Smerovacia tabuľka je načítavaná zo súboru (XML súbor konfigurácie pre ANSARouter). Následne môže byť menená dynamickými smerovacími protokolmi. Aktuálna smerovacia tabuľka sa dá zobraziť v priebehu simulácie. Výstup a jeho položky zodpovedajú zobrazeniu smerovacej tabuľky na Cisco smerovačoch.

Tento modul nemá žiadne vstupné ani výstupné brány. Celá funkcionalita je prístupná cez členské funkcie C++ triedy.

4.1.9 NotificationBoard

Modul NotificationBoard umožňuje informovať ostané moduly o udalostiach (zmenách), ktoré nastali v simulačnom modeli. Udalosti sa môžu týkať zmien v smerovacej tabuľke, zmien konfigurácie rozhraní, zmien kanálu bezdrôtovej siete, či pozície uzlu.

Modul je prístupný priamo cez volanie C++ metód (neprebieha výmena správ). Jednotlivé moduly môžu reagovať na zvolený typ zmeny. Keď daná zmena nastane, patričný modul informuje NotificationBoard, a ten rozdistribuuje informáciu všetkým ostatným zainteresovaným modulom.

4.1.10 NotificationBoard

Modul InterfaceStateManager vytvára prostredníka pre modul ScenarioManager, ktorý sa stará o riadenie priebehu simulácie. InterfaceStateManager prijíma príkazy od modulu ScenarioManager a po ich interpretácii vykoná potrebnú akciu. Typickým príkazom môže byť vypnutie niektorého z rozhraní.

Modul nemá žiadne vstupné ani výstupné brány. Celá funkcionalita je prístupná cez členské funkcie C++ triedy z modulu ScenarioManager.

4.2 Modul OutputQueue

Modul OutputQueue je miesto, kde sú sústredené aktuálne implementované nástroje pre simulovanie kvality služieb IP sietí v rámci v prostredia INET Framework. Je kľúčovým aj z pohľadu ďalšieho rozširovania možností simulácie.

Z pohľadu implementácie samotný modul OutputQueue neobsahuje žiadnu QoS funkcionalitu. Definuje len rozhranie pre komunikáciu, ktoré potom dedia moduly už s konkrétnou implementovanou funkciou. Modul je vždy súčasťou modulu fyzického rozhrania, a to buď Ethernetového rozhrania alebo rozhrania PPP. Obsahuje dve komunikačné brány, jednu vstupnú a druhú výstupnú. Vstupná brána je prepojená na výstupnú bránu modulu sieťovej vrstvy NetworkLayer a výstupná brána je prepojená vstupnú bránu modulu fyzickej vrstvy. Tok paketov teda prechádza zo sieťovej vrstvy do modulu OutputQueue a potom do modulu fyzickej vrstvy. Opačným smerom pakety cez modul OutputQueue neputujú, ale idú priamo z modulu fyzickej vrstvy do modulu sieťovej vrstvy.

Pakety prichádzajúce zo sieťovej vrstvy do modulu OutputQueue sú buď automaticky preposlané na fyzickú vrstvu, pokiaľ nie je aktuálne vysielaný žiadny paket na linku, alebo sú pakety dočasne uložené v rámci modulu a hneď ako sa uvoľní linka, tak modul fyzického rozhrania si od modulu OutputQueue vyžiada paket na odoslanie.

O spôsobe ukladania a metóde výberu paketov rozhoduje už konkrétna implementácia modulu OutputQueue. Každá z implementácií musí definovať minimálne dva povinné funkcie - *enqueue()* a *dequeue()*. *Enqueue()* definuje ako má byť naložené s paketom vchádzajúcim do modulu cez vstupnú branú a *dequeue()* určuje, ktorý paket má byť poslaný fyzickej vrstve, keď si oň požiada.

Aktuálne v INET Framework existujú tri implementácie pre modul OutputQueue:

- DropTailQueue
- DropTailQoSQueue
- REDQueue

4.2.1 DropTailQueue

Modul je definovaný v súbore *src/networklayer/queue/DropTailQueue.ned*. Ide o implementáciu základnej FIFO fronty. Jediným konfigurovateľným parametrom je dĺžka fronty – *frameCapacity*. Tento parameter je možné nastaviť v konfiguračnom súbore *omnetpp.ini*, implicitne je dĺžka fronty nastavená na 100 paketov.

Funkcia *enqueue()* má jednoduchú implementáciu. Vstupujúci paket je uložený do vytvorenej fronty, pokiaľ nie je prekročená maximálna dĺžka fronty. V prípade, že je fronta plná, tak je paket zahodený. Funkcia *dequeue()* vráti prvý paket z fronty, pokiaľ nie je fronta prázdna. V opačnom prípade vracia *NULL*.

4.2.2 DropTailQoSQueue

Modul je definovaný v súbore *src/networklayer/queue/DropTailQoSQueue.ned* a implementuje frontu s prioritným plánovaním. Konkrétne obsahuje dve fronty, jednu pre prioritné pakety a druhú pre všetky ostatné neprioritné pakety. Modul má dva konfigurovateľné parametere: *frameCapacity* – pre definovanie dĺžky každej z front a *classifierClass* – pre určenie klasifikátora určujúceho, ktoré pakety budú zaradené do prioritnej fronty. Nastavenie týchto parametrov je realizovateľné v konfiguračnom súbore omnetpp.ini.

Aktuálne existuje implementácia iba jedného statického klasifikátora *BasicDSCPClassifier*. Tento klasifikátor určuje za prioritné všetky pakety, ktorých 6-bitová DSCP hodnota je v tvare 1XXXX0, kde X je ľubovoľná bitová hodnota (1 alebo 0). Spôsob klasifikácie je pevne daný implementáciou a nie je možné ho konfiguračne meniť. Všetky ostatné pakety sú klasifikované do neprioritnej fronty.

V implementácii funkcie *enqueue()* sa najprv zavolá funkcia klasifikácie. Klasifikátor vracia ID fronty, do ktorej má byť paket zaradený. Pokiaľ je aktuálna dĺžka fronty menšia než maximálna, tak je paket vložený do fronty. V prípade, že je fronta plná, tak je paket zahodený. Pri funkcii *dequeue()* dochádza k výberu paketu z dvoch možných front. Vždy je preferovaný paket z prioritnej fronty, až keď je prioritná fronta prázdna, tak funkcia vracia prvý paket z neprioritnej fronty. Pokiaľ sú obe fronty prázdne, tak funkcia vracia *NULL*.

4.2.3 REDQueue

Modul je definovaný v súbore *src/networklayer/queue/REDQueue.ned*. Ako aj názov napovedá, implementuje FIFO frontu s technikou predchádzania zahlteniu RED (Random Early Detection). Nezohľadňuje dôležitosť jednotlivých paketov, napr. podľa hodnoty DSCP alebo

IP precedence a aplikuje rovnaké politiky zahadzovania paketov pre všetky toky. Modul obsahuje 5 parametrov nastaviteľných v konfiguračnom súbore *omnetpp.ini*. Ide o nasledovné parametre:

- *wq* – určuje, ako výrazne ovplyvňuje aktuálna dĺžka fronty hodnotu priemernej dĺžky fronty. Implicitná hodnota je 0,002.
- *minth* – minimálna dĺžka fronty, pri ktorej je aplikovaný mechanizmus RED. Implicitná hodnota 5.
- *maxth* – maximálna dĺžka fronty, pri ktorej je aplikovaný mechanizmus RED. Definuje tiež kapacitu fronty. Pakety sú po prekročení kapacity zahadzované. Implicitná hodnota je 50.
- *maxp* – určuje maximálnu pravdepodobnosť zahodenia paketu mechanizmom RED. Implicitná hodnota je 0,02.
- *pkrate* – určuje očakávaný počet odosielaných paketov za sekundu. Implicitná hodnota je 150.

Celý mechanizmus RED je implementovaný v rámci funkcie *enqueue()* a je riadený nasledovným algoritmom:

```
// Inicializácia:
//    avg <- 0
//    count <- -1
// pre každý príchod paketu
//    spočítaj novú priemernú dĺžku fronty avg:
//        IF fronta nie je prázdna
//            avg <- (1-wq)*avg + wq*q
//        ELSE
//            m <- f(time-q_time)
//            avg <- (1-wq)^m * avg
//    IF minth <= avg < maxth
//        count++
//        spočítaj pravdepodobnosť pa:
//            pb <- maxp*(avg-minth) / (maxth-minth)
//            pa <- pb / (1-count*pb)
//        s pravdepodobnosťou pa:
//            označ prichádzajúci paket
//            count <- 0
//    ELSE IF maxth <= avg
//        označ prichádzajúci paket
//        count <- 0
//    ELSE count <- -1
// keď sa fronta vyprázdni
//    q_time <- time
//
// Uložené premenné:
//    - avg: priemerná dĺžka fronty
//    - q_time: čas od kedy je fronta prázdna
//    - count: počet paket od naposledy označeného paketu
//
// Fixné parametre:
//    - wq: váha aktuálnej fronty
//    - minth: minimálny threshold pre RED
//    - maxth: maximálny threshold pre RED
```

```
//    - maxp: maximálna pravdepodobnosť zahodenia
//
// Ostatné:
//    - pa: aktuálna pravdepodobnosť označovania
//    - q: aktuálna dĺžka fronty
//    - time: aktuálny čas
//    - f(t): lineárna funkcia času t
```

Algoritmom označené pakety sú zahodené. Pakety bez označenia sú vložené do fronty.

Implementácia funkcie *dequeue()* je podobná ako v module DropTailQueue. Pokiaľ nie je fronta prázdna, je vrátený vždy prvý paket z fronty, v opačnom prípade vracia *NULL*.

Kapitola 5

Implementácia rozšírení INET Framework v oblasti QoS

Z analýzy možností simulácie QoS v prostredí INET Framework z kapitoly 4 vyplýva, že kvalita a množstvo QoS nástrojov sú aktuálne značne obmedzené. Z troch existujúcich modulov front je iba jeden schopný nejakým spôsobom uprednostňovať určitú skupinu paketov. Avšak ani v tomto prípade nejde o nástroj, ktorý by bol flexibilný a schopný simulácie zodpovedajúcej nástrojom dostupným na reálnych hardwarových smerovačoch. Najmä z pohľadu klasifikácie nie možné parametrizovať rozhodnutie, ktoré pakety majú dostať prioritné zaobchádzanie. Celkovo spôsob konfigurácie modulov je neprijateľný pre projekt ANSA, lebo ich konfigurácia sa načítava z inicializačného súboru *omentpp.ini*. Cieľom je, aby konfigurácia bola načítavaná zo XML súboru, ktorý môže byť vygenerovaný z konfiguračného súboru reálneho smerovača.

Na základe vyhodnotenia týchto nedostatkov som sa rozhodol v rámci práce implementovať 4 typy štandardných front. Konkrétne ide o implementáciu FIFO, WFQ, PQ a CQ v module, ktorý som nazval AnsaQosSystem.

V simulácii QoS je podstatné vyhodnotenie, aký majú nastavené QoS politiky dopad na sieťové prostredie a prenášané dáta aplikácií. K tomu je potrebné tieto dáta generovať a výsledok prenosu následne analyzovať. Aktuálne v prostredí INET Framework neexistuje modul, ktorý by bol schopný túto činnosť použiteľne realizovať. Preto som v rámci mojej práce implementoval generátor dátových tokov s pokročilou analýzou. Modul, v ktorom som túto funkcionalitu implementoval, som nazval TrafGen.

5.1 Modul AnsaQosSystem

Modul AnsaQosSystem implementuje fronty FIFO, WFQ, PQ a CQ v prostredí INET Framework. Jeho definícia sa nachádza v súbore *src/ansa/qos/AnsaQosSystem.ned*. Modul dedí vlastnosti modulu OutputQueue, vďaka čomu je integrovateľný do ktoréhokoľvek modulu fyzického rozhrania v INET Framework. Má jednu vstupnú a jednu výstupnú bránu. Vstupná brána je prepojená na výstupnú bránu modulu sieťovej vrstvy NetworkLayer a výstupná brána je prepojená na vstupnú bránu modulu fyzickej vrstvy. Modul obsahuje jeden povinný parameter – *configFile*, pre zadanie názvu XML súboru s konfiguráciou smerovača, ktorého je modul súčasťou.

V rámci implementácie modulu sú vytvorené nasledujúce triedy definované v jazyku C++:

- ANSAQOS::QueueConfig – trieda pre uchovanie konfigurácie a aktuálneho stavu fronty. Medzi hlavné parametre, ktoré obsahuje, patrí typ fronty (FIFO, WFQ,PQ,CQ), maximálna a aktuálna veľkosť fronty, naposledy obsluhovaná podfronta a implicitná podfronta.
- ANSAQOS::SNPacket – trieda definuje formát uloženia paketu vo fronte. Konkrétne pre potreby WFQ je nutné spolu s paketom udržiavať aj jeho sekvenčné číslo.
- ANSAQOS::SubQueue – trieda reprezentuje jednu podfrontu a definuje operácie nad ňou. Obsahuje vektor uložených paketov a objekt klasifikátora. Okrem toho sú uchovávané štatistické informácie ako počet zahodených paketov, aktuálna a maximálna veľkosť.
- ANSAQOS::Classifier – trieda definuje rozhranie pre konkrétne implementácie klasifikátora. Vďaka tomuto môže mať každá podfronta klasifikátor dynamicky načítavaný z konfiguračného súboru.
- AnsaQosSystem – je trieda implementujúca samotný modul. Obsahuje objekt definujúci jeho konfiguráciu a vektor podfront. Implementuje metódy načítania konfigurácie a vkladania respektíve výberu paketu z fronty.

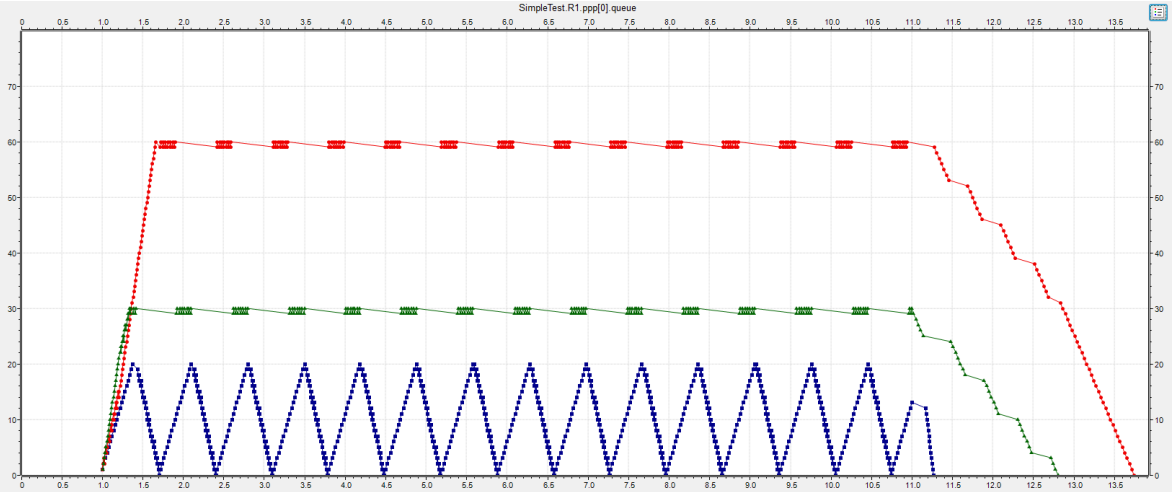
Základný princíp funkcie modulu je možné zhrnúť takto:

- Hneď pri spustení simulácie je načítaná konfigurácia modulu. Z parametra *configFile* je získaný názov XML súboru obsahujúceho konfiguráciu. Je očakávaná nasledujúca štruktúra XML dokumentu:

```
<Routers>
  <Router id="...">
    <Interfaces>
      <Interface name="...">
        <OutputQueue type="...">
          ...
        </OutputQueue>
      </Interface>
    </Interfaces>
  </Router>
</Routers>
```

- Na základe ID smerovača, ktorého je modul súčasťou, je nájdený správny *Router* element. Následne podľa mena rozhrania, ktorého je modul súčasťou, získa *Interface* element. V rámci *Interface* elementu pre svoju konfiguráciu očakáva element *OutputQueue*. Atribút *type* potom rozhodne aký typ fronty bude načítaný. Konkrétna konfigurácia je závislá na typu fronty. V tomto kroku sú vytvorené všetky potrebné podfronty a ich klasifikátory. Pokiaľ element *OutputQueue* nie je definovaný, je implicitne použitá FIFO fronta.
- Na prichádzajúci paket modul reaguje funkciou *enqueue()*, ktorá spracuje paket podľa typu fronty.
- V prípade, že modul fyzického rozhrania môže odoslať ďalší paket, tak zavolá funkciu *dequeue()*. Funkcia vráti *NULL*, pokiaľ modul neobsahuje žiadny paket. V opačnom prípade sa na základe typu fronty rozhodne o tom, ktorý paket bude vybraný.

- Pre všetky typy front sú do vektorového súboru VEC ukladané štatistiky o celkovej dĺžke fronty a celkovému počtu zahodených paketov. Pre PQ a CQ sú tieto štatistiky ukladané aj pre jednotlivé podfronty. V prípade WFQ sa podfronty vytvárajú dynamicky a existujú len po dobu, kým obsahujú nejaký uložený paket. Preto nie je možné nejako rozumne uchovávať štatistky podfront aj pre WFQ. Výsledky simulácie sa potom dajú v prostredí OMNeT++ ďalej spracovávať a na základe VEC súborov vygenerovať grafy. Obrázok 5.1 zobrazuje graf priebežnej dĺžky podfront PQ, ako boli modulom AnsaQosSystem zaznamenané počas simulácie.



Obr. 5.1: Ukážka grafu priebežnej veľkosti podfront v PQ

5.1.1 Implementácia klasifikátorov

Do každej podfronty musí byť priradený klasifikátor. Každý konkrétny klasifikátor musí zdediť rozhranie od triedy *ANSAQOS::Classifier*. V rámci tejto práce som implementoval nasledujúce klasifikátory:

- MatchNoneClassifier – neklasifikuje žiadny paket.
- MatchAnyClassifier – klasifikuje každý paket.
- PRECClassifier – klasifikuje pakety s danou hodnotou IP precedence v IP hlavičke. V rámci jedného klasifikátora je možné špecifikovať viac hodnôt IP precedence, potom je pri klasifikácii aplikovaná logická operácia OR.
- DSCPClassifier – klasifikuje pakety s danou hodnotou DSCP v IP hlavičke. V rámci jedného klasifikátora je možné špecifikovať viac hodnôt DSCP, potom je pri klasifikácii aplikovaná logická operácia OR.
- ACLClassifier – klasifikuje pakety na základe ACL. Aby bol paket klasifikovaný musí byť v ACL vyhodnotený s akciou "permit". V rámci jedného klasifikátora je možné špecifikovať viac ACL, potom je pri klasifikácii aplikovaná logická operácia OR.

- WFQClassifier – klasifikuje paket na základe toku. Tok je definovaný šesticou: zdrojová a cieľová IP adresa, zdrojový a cieľový port, protokol a IP precedence.

Pre klasifikáciu paketov pomocou ACL som vytvoril modul AclContainer (definovaný v *src/ansa/acl/AclContainer.ned*), ktorý je vložený ako podmodul do modulu smerovača. Modul vychádza z implementácie modulu ACL pre filtrovanie paketov [5]. Princíp špecifikácie ACL v konfiguračnom súbore ostal zachovaný, ide však o zovšeobecnenie tohto modulu pre možnosť použitia ACL aj pre iné aplikácie, ako je filtrovanie paketov. Implementovaný modul nemá vstupné ani výstupné brány. K jeho funkciám sa dá dostať pomocou metódy *AclContainerAccess()*, podobným spôsobom, ako je to v prípade modulov RoutingTable alebo InterfaceTable. Modul pri inicializácii načíta všetky nakonfigurované ACL v rámci daného smerovača. Následne umožňuje volanie funkcie *matchPacketToAcl(acl, paket)*, kde parameter *acl* určuje meno ACL, voči ktorému sa má paket klasifikovať a parameter *paket* obsahuje paket určený ku klasifikácií. Funkcia vráti true pokiaľ výsledná akcia je "permit". Obrázok 5.2 zobrazuje obsah modulu AclContainer v prípade konfigurácie smerovača s jedným ACL, definovaným nasledujúcou XML syntaxou:

```
<Router id="...">
  <ACLs>
    <ACL no="110">
      <entry seq_no="10">
        <action>permit</action>
        <IP_src>172.16.100.2</IP_src>
        <IP_dst>172.16.200.0</IP_dst>
        <WC_src>0.0.0.0</WC_src>
        <WC_dst>0.0.0.255</WC_dst>
        <port_op_src></port_op_src>
        <port_op_dst>eq</port_op_dst>
        <port_beg_src></port_beg_src>
        <port_end_src></port_end_src>
        <port_beg_dst>53</port_beg_dst>
        <port_end_dst></port_end_dst>
        <protocol>udp</protocol>
        <orig>access-list 110 permit udp host 172.16.100.2 172.16.200.0 0.0.0.255 eq 53</orig>
      </entry>
    </ACL>
  </ACLs>
</Router>
```

5.1.2 Implementácia FIFO

Samotný návrh modulu AnsaQosSystem počíta s existenciou podfront. Metóda FIFO však žiadne podfronty neobsahuje, preto v implementácii je pre FIFO vytvorená jedna podfronta, nad ktorou je algoritmus FIFO realizovaný.

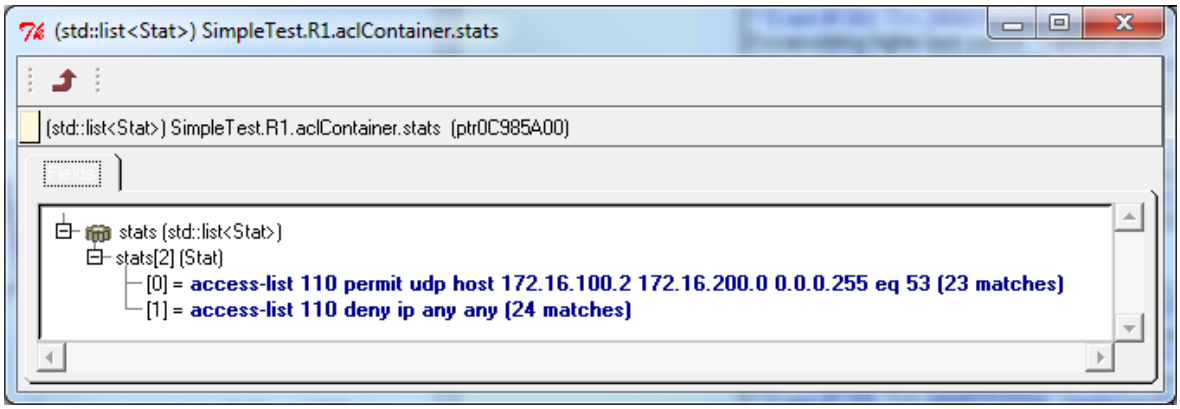
Funkcia *enqueue()* v prípade FIFO vkladá paket do jednej existujúcej podfronty a pokiaľ nie je plná, tak je paket do nej vložený. Nedochádza k žiadnej klasifikácií.

Funkcia *dequeue()* vyberie a vráti vždy prvý paket podfronty.

Konfigurácia FIFO očakáva nasledovnú štruktúru elementu *OutputQueue*:

```
<OutputQueue type="FIFO">
  <HoldQueueLength>50</HoldQueueLength>
</OutputQueue>
```

Kde element *HoldQueueLength* určuje dĺžku fronty. Môže byť aj vynechaný, potom je použitá implicitná hodnota 75 paketov.



Obr. 5.2: ACL v module AclContainer

Obrázok 5.3 zobrazuje parametre a štatistiky, ktoré sa u FIFO fronty dajú v priebehu simulácie sledovať. Parameter *strategy* je v tomto prípade zobrazený ako *fifo* a medzi najdôležitejšie pozorovateľné štatistiky patrí:

- *size* – aktuálna veľkosť fronty
- *dropped* – počet zahodených paketov
- *max* – maximálna veľkosť fronty

Class	Name	Info
cPar	configFile	"RoutersConfig-FIFO.xml"
cGate	in	<-- <parent>.netwIn, ned.IdealChannel
cGate	out	--> ppp.netwIn
cOutVector	FIFO Drops	received 764 values, stored 764
cOutVector	FIFO Length	received 421 values, stored 421
i	packetRequested	0
i	numQueueReceived	1000
i	numQueueDropped	764
ANSAQOS::QueueConfig	QueueInfo	Strategy: fifo; Queue: 49/764/50 (size/dropped/max)
std::vector<ANSAQOS::SubQueue>	subQueues	size=1

Obr. 5.3: Parametre a štatistiky FIFO sledovateľné v priebehu simulácie

5.1.3 Implementácia WFQ

V metóde WFQ sa podfronty vytvárajú dynamicky, preto pri inicializácii modulu nie je vytvorená žiadna podfronta. Podfronta sa vytvorí pri príchode prvého paketu daného toku a zanikne hneď po jej vyprázdnení. Podfronte sa automaticky priradí klasifikátor toku na základe hodnôt hlavičiek prvého paketu.

Funkcia *enqueue()* sa snaží najprv klasifikovať paket do jednej z existujúcich podfront, pokiaľ sa to nepodarí, tak vytvorí novú podfrontu (pokiaľ nie je dosiahnutý limit počtu podfront). Paket je v podfronte uložený v objekte triedy *ANSAQOS::SNPacket*, ktorý okrem samotného paketu obsahuje aj sekvenčné číslo. Sekvenčné číslo je paketu vypočítané na základe vzorca zo sekcie 2.4.2.

Funkcia *dequeue()* prehľadá všetky podfronty a nájde paket s najnižším sekvenčným číslom, ktorý vyberie a vráti. Pokiaľ vybratie paketu znamená vyprázdnenie podfronty, tak dôjde k jej zrušeniu.

Konfigurácia WFQ očakáva nasledovnú štruktúru elementu *OutputQueue*:

```
<OutputQueue type="WFQ">
  <HoldQueueLength>1000</HoldQueueLength>
  <CDT>64</CDT>
  <DynamicQueues>256</DynamicQueues>
</OutputQueue>
```

Element *HoldQueueLength* určuje celkové množstvo paketov, ktoré sa môže vo WFQ nachádzať. *CDT* určuje maximálnu dĺžku podfronty a *DynamicQueues* stanovuje maximálny počet podfront. Pokiaľ niektorý z týchto parametrov nie je uvedený, tak je požitá implicitná hodnota ($HoldQueueLength = 1000$, $CDT = 64$, $DynamicQueues = 256$).

Obrázok 5.4 zobrazuje parametre a štatistiky, ktoré sa dajú u WFQ fronty v priebehu simulácie sledovať. Parameter *strategy* je v tomto prípade zobrazený ako *weighted fair* a medzi najdôležitejšie pozorovateľné štatistiky patrí:

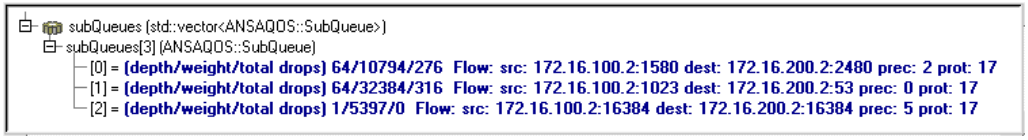
- *Queue size* – aktuálny počet paketov vo všetkých podfrontách
- *Queue max total* – maximálny limit paketov, ktoré môže WFQ fronta obsahovať
- *Queue threshold* – maximálny limit paketov pre jednu podfrontu
- *Queue drops* – celkový počet zahodených paketov
- *Conversations active* – aktuálny počet aktívnych tokov
- *Conversations max active* – maximálny dosiahnutý počet aktívnych tokov
- *Conversations max total* – maximálny limit počtu aktívnych tokov

Na obrázku 5.5 sú zobrazené parametre a štatistiky podfront jednotlivých tokov.Okrem informácie o samotnom toku sú zaujímavé tieto štatistické informácie:

- *depth* – aktuálna veľkosť podfronty
- *weight* – váha podfronty
- *total drops* – počet zahodených paketov podfronty

Class	Name	Info
cPar	configFile	"RoutersConfig-WFQ.xml"
cGate	in	<-- <parent>.netwIn, ned.IdealChannel
cGate	out	--> ppp.netwIn
cOutVector	WFQ Drops	received 592 values, stored 592
cOutVector	WFQ Length	received 685 values, stored 685
i	packetRequested	0
i	numQueueReceived	1000
i	numQueueDropped	592
ANSAQOS::QueueConfig	QueueInfo	Strategy: weighted fair; Queue: 129/1000/64/592 (size/max total/threshold/drops); Conversations: 3/3/256 (active/max active/max total)
std::vector<ANSAQOS::SubQueue>	subQueues	size=3

Obr. 5.4: Parametre a štatistiky WFQ sledovateľné v priebehu simulácie



Obr. 5.5: Parametre a štatistiky dynamických podfront WFQ

5.1.4 Implementácia PQ

V metóde PQ sú pri inicializácií modulu vytvorené 4 podfronty (High, Medium, Normal, Low), ktoré sú do vektora podfront zaradené podľa priority.

Funkcia *enqueue()* sa snaží na základe klasifikácie vložiť paket do jednej z podfront. Pokiaľ ku klasifikácii nedôjde, tak je paket vkladaný do implicitnej podfronty. Výsledok vkladania je závislý na tom, či daná podfronta dosiahla maximálnu dĺžku.

Funkcia *dequeue()* prechádza podfronty podľa priority a vyberie paket z prvej neprázdnej podfronty.

Konfigurácia PQ očakáva nasledovnú štruktúru elementu *OutputQueue*:

```
<OutputQueue type="PQ">
  <PriorityQueueList>1</PriorityQueueList>
</OutputQueue>
```

Kde element *PriorityQueueList* určuje identifikátor konfigurácie PQ v rámci konfigurácie smerovača. Konfigurácia parametrov PQ prebieha pomocou "priority listov"nasledovným spôsobom:

```
<Router id="...">
  <PriorityQueueLists>
    <PriorityQueueList id="1">
      <DefaultQueue>Low</DefaultQueue>
      <HighQueue>
        <Classifier type="DSCP">
          <value>ef</value>
        </Classifier>
      </HighQueue>
      <MediumQueue>
        <Lenght>30</Lenght>
        <Classifier type="PREC">
          <value>2</value>
        </Classifier>
      </MediumQueue>
      <NormalQueue>
        <Lenght>30</Lenght>
        <Classifier type="ACL">
          <acl>110</acl>
        </Classifier>
      </NormalQueue>
      <LowQueue>
        <Lenght>60</Lenght>
      </LowQueue>
    </PriorityQueueList>
  </PriorityQueueLists>
</Router>
```

Každý priority list má svoj identifikátor. Element *DefaultQueue* určuje implicitnú podfrontu. Pokiaľ nie je špecifikovaný, tak je ako implicitná podfronta nastavená podfronta

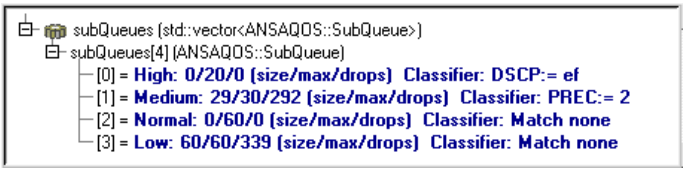
Normal. Každej podfronte je možné elementom *Lenght* nastaviť jej dĺžku. Pokiaľ nie je dĺžka explicitne špecifikovaná, je nastavená implicitná dĺžka (High = 20, Medium = 40, Normal = 60, Low = 80). Taktiež je možné pomocou elementu *Classifier* nastaviť klasifikátor. Podporovaná je klasifikácia na základe DSCP, IP precedence alebo ACL. V prípade, že klasifikátor nie je uvedený, tak podfronte bude pridelený klasifikátor, ktorý neklasifikuje žiadny paket.

Obrázok 5.6 zobrazuje parametre a štatistiky, ktoré sa u PQ fronty dajú v priebehu simulácie sledovať. Parameter *strategy* je v tomto prípade zobrazený ako *priority-list id* a parameter *default queue* zobrazuje definovanú implicitnú podfrontu. Na obrázku 5.7 sú zobrazené parametre a štatistiky podfront PQ. Význam jednotlivých položiek je nasledovný:

- *size* – aktuálna veľkosť podfronty
- *max* – maximálna veľkosť podfronty
- *drops* – počet zahodených paketov podfronty
- *classifier* – typ klasifikátora podfronty a jeho hodnota

Class	Name	Info
cPar	configFile	"RoutersConfig-PQ.xml"
cGate	in	<-- <parent>.netwIn, ned.IdealChannel
cGate	out	--> ppp.netwIn
cOutVector	PQ Drops	received 631 values, stored 631
cOutVector	PQ Length	received 647 values, stored 647
i	packetRequested	0
i	numQueueReceived	1000
i	numQueueDropped	631
ANSAQOS::QueueConfig	QueueInfo	Strategy: priority-list 10 (Default queue: Low)
std::vector<ANSAQOS::SubQueue>	subQueues	size=4

Obr. 5.6: Parametre a štatistiky PQ sledovateľné v priebehu simulácie



Obr. 5.7: Parametre a štatistiky podfront PQ

5.1.5 Implementácia CQ

V metóde CQ je pri inicializácií modulu vytvorených 16 podfront, ktoré sú do vektora podfront radené podľa ID. Každá podfronta má definovanú hodnotu "byte count", ktorá určuje maximálny počet bajtov vybraných z podfronty pri jednom prechode round-robin.

Podobne ako v prípade PQ sa funkcia *enqueue()* snaží na základe klasifikácie vložiť paket do jednej z podfront. Pokiaľ ku klasifikácii nedôjde, tak je paket vkladaný do implicitnej podfronty. Výsledok vkladania je závislý na tom, či daná podfronta dosiahla maximálnu dĺžku.

Funkcia *dequeue()* prechádza podfronty spôsobom round-robin. Je udržiavaná informácia o aktuálnej podfronte. Z aktuálnej podfronty vyberie paket, pokiaľ počet vybraných bajtov nepresahuje byte count danej podfronty a podfronta je neprázdna. V opačnom prípade sa ukazovateľ aktuálnej podfronty posúva na ďalšiu neprázdnu podfrontu a výber sa opakuje.

Konfigurácia CQ očakáva nasledovnú štruktúru elementu *OutputQueue*:

```
<OutputQueue type="CQ">
  <CustomQueueList>1</CustomQueueList>
</OutputQueue>
```

Kde element *CustomQueueList* určuje identifikátor konfigurácie CQ v rámci konfigurácie smerovača. Konfigurácia parametrov CQ prebieha pomocou "custom listov"nasledovným spôsobom:

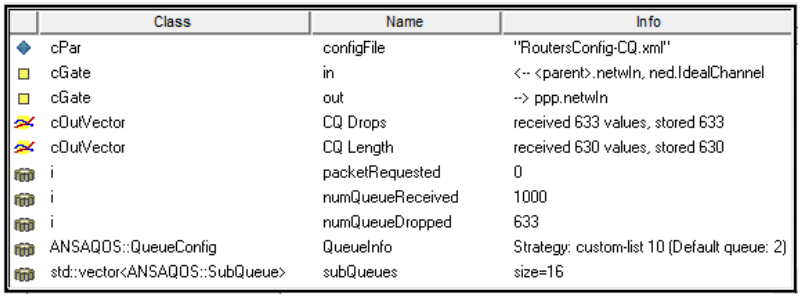
```
<Router>
  <CustomQueueLists>
    <CustomQueueList id="1">
      <DefaultQueue>2</DefaultQueue>
      <CustomQueue id="1">
        <ByteCount>3000</ByteCount>
        <Classifier type="DSCP">
          <value>af41</value>
        </Classifier>
      </CustomQueue>
      <CustomQueue id="2">
        <Lenght>60</Lenght>
        <ByteCount>2000</ByteCount>
        <Classifier type="PREC">
          <value>3</value>
        </Classifier>
      </CustomQueue>
      <CustomQueue id="3">
        <Lenght>30</Lenght>
        <Classifier type="ACL">
          <acl>110</acl>
        </Classifier>
      </CustomQueue>
    </CustomQueueList>
  </CustomQueueLists>
</Router>
```

Každý custom list má svoj identifikátor. Element *DefaultQueue* určuje implicitnú podfrontu. Pokiaľ nie je špecifikovaný, tak je ako implicitná podfronta nastavená podfronta s ID 1. Každej podfronte je možné elementom *Lenght* nastaviť jej dĺžku. Pokiaľ nie je dĺžka explicitne špecifikovaná, je nastavená implicitná dĺžka 20 paketov. Ďalej je možné pomocou elementu *ByteCount* špecifikovať počet bajtov, ktoré podfronta uvoľní pri jednom round-robin prechode. Implicitná hodnota je 1500 bajtov. Každej podfronte je možné pomocou elementu *Classifier* nastaviť klasifikátor. Podporovaná je klasifikácia na základe DSCP, IP precedence alebo ACL. V prípade, že klasifikátor nie je uvedený, tak podfronte bude pridelený klasifikátor, ktorý neklasifikuje žiadny paket.

Obrázok 5.8 zobrazuje parametre a štatistiky, ktoré sa u PQ fronty dajú v priebehu simulácie sledovať. Parameter *strategy* je v tomto prípade zobrazený ako *custom-list id* a parameter *default queue* zobrazuje definovanú implicitnú podfrontu. Na obrázku 5.9 sú zobrazené parametre a štatistiky podfront PQ. Význam jednotlivých položiek je nasledovný:

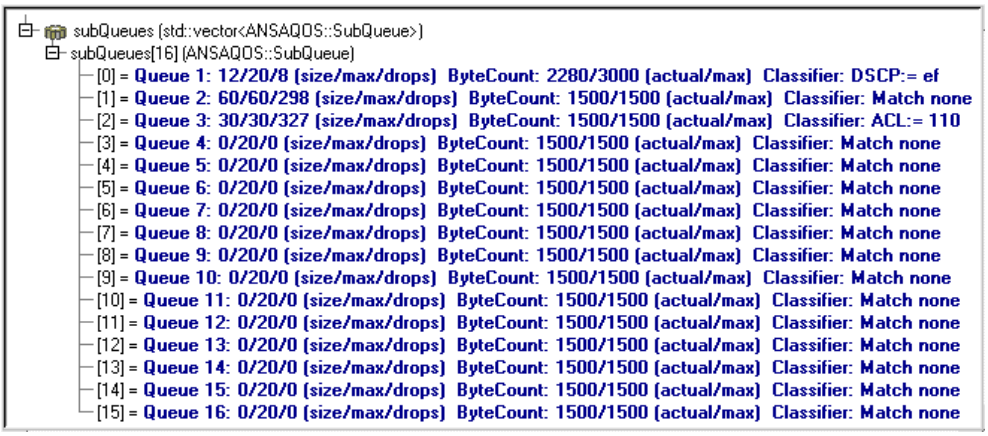
- *Queue size* – aktuálna veľkosť podfronty

- *Queue max* – maximálna veľkosť podfronty
- *Queue drops* – počet zahodených paketov podfronty
- *ByteCount actual* – aktuálne množstvo bajtov, ktoré má dovolené podfronta vyslať
- *ByteCount max* – maximálne množstvo bajtov podfonty pre jeden prechod metódou round-robin
- *classifier* – typ klasifikátora podfronty a jeho hodnota



Class	Name	Info
cPar	configFile	"RoutersConfig-CQ.xml"
cGate	in	<-- <parent>.netwIn, ned.IdealChannel
cGate	out	--> ppp.netwIn
cOutVector	CQ Drops	received 633 values, stored 633
cOutVector	CQ Length	received 630 values, stored 630
i	packetRequested	0
i	numQueueReceived	1000
i	numQueueDropped	633
ANSAQOS::QueueConfig	QueueInfo	Strategy: custom-list 10 (Default queue: 2)
std::vector<ANSAQOS::SubQueue>	subQueues	size=16

Obr. 5.8: Parametre a štatistiky CQ sledovateľné v priebehu simulácie



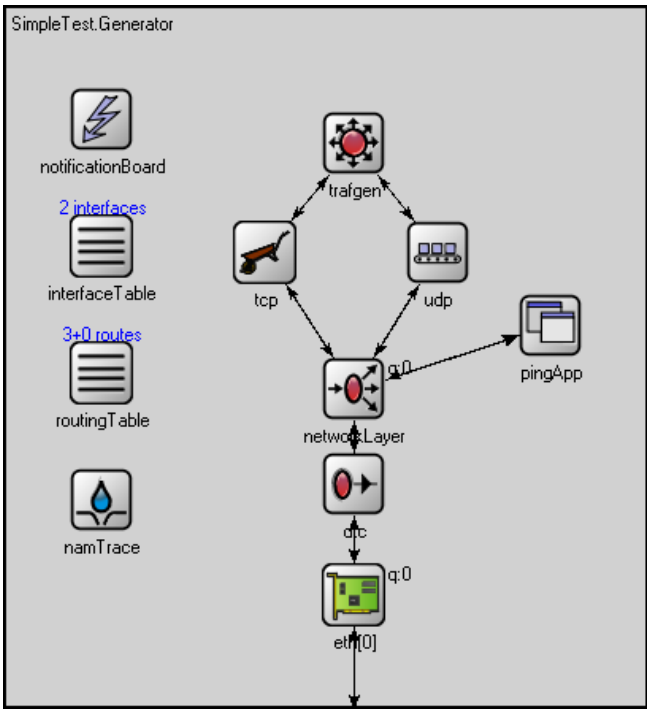
Obr. 5.9: Parametre a štatistiky podfront CQ

5.2 Modul TrafGen

Z pohľadu simulácie QoS je potrebné mať možnosť generovania dátových tokov s rozmanitou možnosťou špecifikácie parametrov. Jedine týmto spôsobom je potom možné sledovať, ako sú pakety jednotlivých tokov klasifikované a plánované v rámci rôznych plánovacích algoritmov a tiež, ako sa vyvíja priebežná dĺžka jednotlivých front. Práve k tomuto určeniu som v rámci tejto práce vytvoril modul, ktorý som nazval TrafGen. Modul TrafGen umožňuje generovať dátové toky prenášané pomocou transportných protokolov TCP alebo UDP. Pre prenášané pakety sú vyhodnocované základné štatistiky týkajúce sa stratovosti, oneskorenia a rozptylu oneskorenia.

5.2.1 Princíp funkcie modulu TrafGen

Modul TrafGen má dve vstupné a dve výstupné brány. Jedna dvojica slúži pre pripojenie k modulu TCP a druhá pre pripojenie k modulu UDP. Ide o moduly transportnej vrstvy, ktoré sú súčasťou implementácie INET Framework. Modul Trafgen teda v princípe môže byť súčasťou modulu StandardHost, ktorý reprezentuje užívateľský počítač. Pre jednoduchosť použitia som vytvoril vlastný modul ANSATrafGenHost, ktorý je odvodený od modulu StandardHost a obsahuje už pripojený modul TrafGen. Vnútorná štruktúra modulu ANSATrafGenHost je zobrazená na obrázku 5.10.



Obr. 5.10: Vnútorná štruktúra modulu ANSATrafGenHost

Pre korektnú funkčnosť modul ANSATrafGenHost obsahuje oproti modulu StandardHost ešte jednu komponentu navyše. Jedná sa o modul DscpTtlCorrector vložený medzi modul sieťovej vrstvy a modul fyzického rozhrania, ktorý slúži ku korekcii hodnôt DSCP a TTL v IP hlavičkách generovaných paketov. Je to implementované týmto spôsobom z dôvodu nemožnosti ovplyvňovať hodnoty v IP hlavičke priamo z aplikačnej vrstvy, teda z modulu TrafGen. Modul DscpTtlCorrector nastavuje hodnotu DSCP a TTL podľa definície toku zistenej priamo z modulu TrafGen.

Pri simulácii sú v topológii typicky dva a viac moduly ANSATrafGenHost. Pre každý tok je jeden modul, ktorý daný tok generuje a jeden modul, čo ten istý tok analyzuje. Modul môže súčasne niektoré toky generovať a iné zase analyzovať.

Po spustenú simulácie si modul načíta svoju konfiguráciu zo XML súboru obsahujúceho definíciu všetkých tokov. Po úspešnom načítaní modul zistí, ktoré toky bude generovať, a ktoré bude analyzovať. V konfigurácii môžu byť aj toky, pre ktoré nebude ani generátor ani analyzátor. To, v akej úlohe bude vystupovať, zisťuje podľa IP adresy svojho ethernetového rozhrania a zdrojovej, respektíve cieľovej IP adresy toku. Modul prejde zoznam všetkých tokov a v prípade, že IP adresa rozhrania je zhodná so zdrojovou IP adresou toku, tak bude

modul daný tok generovať. Naopak, v prípade, že IP adresa rozhrania je zhodná s cieľovou IP adresou toku, tak modul bude daný tok analyzovať. Pri inicializácii musí okrem iného ešte generátor toku získať ukazovateľ do štatistík analyzátora. Potom pri odoslaní každého vygenerovaného paketu môže priamo aktualizovať informáciu o počte odoslaných dát, ktorú môže následne analyzátor využiť pre vytvorenie korektných súhrnných štatistík.

Pre každý tok, v ktorom je modul analyzátorom, vytvorí v závislosti na použitom transportnom protokole, TCP, respektíve UDP soket. Na tomto sokete bude naslúchať. Pri použití transportného protokolu TCP, generátor pred odoslaním prvého paketu inicializuje zostavenie TCP spojenia. V prípade transportného protokolu UDP dochádza iba k vysielaniu paketov na sieť, stavové spojenie sa nevytvára.

Samotné generovanie paketov je plánované časovačmi vo forme zasielania správy samému sebe. Na začiatku sa u každého toku inicializuje časovač na hodnotu definovanú v konfigurácii. Po vypršaní časovača je paket vygenerovaný a vložený na sieť pre ďalší transport. Následne je naplánovaný čas generovania ďalšieho paketu. Konkrétny čas je závislý na použitej aplikácii. Plánovanie končí v prípade, že naplánovaný čas by presahoval dobu generovania toku definovanú v konfigurácii.

Na strane analyzátora je po obdržaní paketu určený tok, do ktorého paket patrí a tomuto toku sú aktualizované štatistiky. Následné paket už nie je nijako spracovávaný a je jednoducho zahodený.

Na aplikačnej úrovni (nad úrovňou transportného protokolu) modul TrafGen definuje jednotný formát generovaných správ. Definícia na úrovni OMNeT++ vyzerá nasledovne:

```
packet TrafGenPacket
{
  string flowId;
  double sentTime;
  string application;
}
```

Kde *flowId* je reťazec s identifikátorom toku, *sentTime* je simulačný čas v dobe odoslania paketu a *application* je reťazec s názvom prenášanej aplikácie.

Z definície paketu je možné vidieť, že k odlíšeniu aplikácie dochádza len na úrovni obsahu reťazca. Žiadna aplikácia nemá na aplikačnej úrovni vlastnú štruktúru paketu, ktorá by zodpovedala reálnej aplikácii. Z pohľadu simulácie QoS reálny obsah aplikačných dát nie je podstatný. Dôležité je, aby aplikácia používala správny transportný protokol. Aby časové rozloženie generovaných paketov zodpovedalo reálnej aplikácii, a tiež, aby veľkosti paketov boli zhodné s reálnou aplikáciou. To znamená, že správanie aplikácie z pohľadu transportu musí zodpovedať realite. Jediným problémom by potom mohla byť klasifikácia na základe aplikácie, ale tá môže byť vyriešená pomocou reťazca aplikácie v generovanom pakete.

Čo sa týka aplikácií, tak ich definícia je riešená modulárne, kde pre každú aplikáciu je možné vytvoriť vlastnú triedu reprezentujúcu správanie danej reálnej aplikácie. Trieda musí dediť od triedy *AITrafGenApplication*. Táto trieda definuje komunikačné rozhranie medzi modulom TrafGen a triedami aplikácií. V rámci implementácie triedy aplikácie je potrebné predefinovať nasledujúce funkcie:

- *loadConfig()* – funkcia, ktorá načíta konfiguráciu aplikácie zo XML súboru.
- *getNextPacketTime()* – funkcia vráti čas, kedy má byť naplánované generovanie ďalšieho paketu.

- *getPacketSize()* – funkcia vráti veľkosť generovaného paketu.
- *getDefaultPort()* – funkcia vráti implicitný port používaný aplikáciou.
- *anotherEncapsulationOverhead()* – funkcia vráti veľkosť hlavičiek nad rámec aplikačných dát, do ktorých môže aplikácia zabaľovať pakety pri prenose (príkladom je veľkosť RTP hlavičky pri VoIP prenose).

Pri načítaní definície toku je podľa názvu aplikácie dynamicky vytvorený jej objekt (musí existovať trieda s daným názvom). Konštruktoru pri vytváraní objektu je ako parameter predložená časť XML, ktorá je určená pre konfiguráciu aplikácie. Teda, každá aplikácia môže mať vlastný spôsob konfigurácie.

V rámci tejto práce som vytvoril dva typy aplikácií. Jedna obecná s názvom *custom*, ktorá umožňuje generovať pakety s vlastnou definíciou časového rozloženia a rozloženia veľkosti. Umožňuje použitie oboch podporovaných transportných protokolov. Druhá aplikácia má názov *voice* a simuluje dátové toky IP telefónie, konkrétne prenosu hlasových paketov protokolom RTP nad transportným protokolom UDP. V rámci tejto aplikácie je možné nastaviť používaný kódek, počet paketov generovaných za sekundu a použitie technológie VAD (Voice Activity Detection), ktorá umožňuje detekciu miest, kedy nie je potrebné hlas prenášať, a tým ušetriť prenosovú kapacitu. Funkcia VAD je implementovaná čisto štatistickým spôsobom. Podľa štatistiky je možné touto technológiou ušetriť 30% prenosovej kapacity linky, a tak pri implementácii generovania vďaka tejto funkcii ostane 30% paketov nevygenerovaných. Konkrétny spôsob a možnosti konfigurácie aplikácií budú prezentované v nasledujúcej sekcii.

5.2.2 Konfigurácia modulu TrafGen

Modul TrafGen očakáva svoju konfiguráciu vo forme XML súboru. V tomto súbore sú definované jednotlivé toky. XML súbor môže byť spoločný pre všetky moduly TrafGen v topológii. Potom toky, pre ktoré daný modul nie je ani generátorom ani analyzátorom, jednoducho ignoruje. Konfigurácia tokov musí mať nasledovnú XML štruktúru:

```
<trafgen>
  <flow id="1">
    <description>UDP Tok 1</description>
    <duration>10.5</duration>
    <start_time>1.5</start_time>
    <ip_header>
      <source_ip>172.16.100.2</source_ip>
      <destination_ip>172.16.200.2</destination_ip>
      <tos>EF</tos>
      <ttl>12</ttl>
      <protocol>udp</protocol>
    </ip_header>
    <transport_header>
      <source_port>1023</source_port>
      <destination_port>53</destination_port>
    </transport_header>
    <application type="...">
      ...
    </application>
  </flow>
  <flow id="2">
    ...
```

```
  </flow>
</trafgen>
```

Jednotlivé elementy XML majú nasledovný význam:

- flow id – jedinečný identifikátor toku (môže to byť aj reťazec znakov)
- description – nepovinný popis toku (reťazec)
- duration – dĺžka generovania toku v sekundách (desatinné číslo)
- start_time – simulačný čas začiatku generovania toku (desatinné číslo)
- source_ip – zdrojová IP adresa toku
- destination_ip – cieľová IP adresa toku
- tos – typ služby (celé číslo z rozsahu 0-255 alebo znakové hodnoty DSCP definované štandardom)
- ttl – životnosť paketu (celé číslo z rozsahu 0-255)
- protocol – transportný protokol (udp/tcp)
- source_port – zdrojový port (celé číslo z rozsahu 0-65536)
- destination_port – cieľový port (celé číslo z rozsahu 0-65536)
- application type – typ použitej aplikácie

XML blok s konfiguráciou aplikácie sa líši podľa typu použitej aplikácie. XML blok konfigurácie aplikácie *custom*:

```
<application type="custom">
  <packets_per_second>100</packets_per_second>
  <time_distribution>constant</time_distribution>
  <packet_size>200</packet_size>
  <size_distribution>constant</size_distribution>
</application>
```

Jednotlivé XML elementy majú nasledovný význam:

- packets_per_second – počet paketov generovaných za sekundu (desatinné číslo)
- time_distribution – časová distribúcia generovania paketov (constant/normal/exponential)
- packet_size – veľkosť generovaných paketov v bajtoch (celé číslo)
- size_distribution – distribúcia veľkosti generovaných paketov (constant/normal/exponential)

XML blok konfigurácie aplikácie *voice*:

```
<application type="voice">
  <codec>g729</codec>
  <packets_per_second>default</packets_per_second>
  <vad>false</vad>
</application>
```

Jednotlivé XML elementy majú nasledovný význam:

- codec – kódek, ktorým je kódovaný hlas do digitálnej podoby (g711/g729)
- packets_per_second – počet paketov generovaných za sekundu (desatinné číslo/default)
- vad – použitie technológie VAD (true/false)

5.2.3 Štatistiky generované modulom TrafGen

Každý modul, ktorý je v pozícii analyzátora, zapisuje pre každý prichádzajúci paket štatistické informácie. Tieto informácie je možné sledovať pribežne v grafickom rozhraní simulácie. Na konci simulácie sú všetky štatistiky zapísané do textového súboru. Súbor je uložený v zložke *result* a jeho názov má tvar *TrafGen-nazov_modulu.txt*, kde *nazov_modulu* je konkrétne pomenovanie modulu TrafGen v simulácii. Obsah vygenerovaného súboru má nasledujúcu podobu:

```
-------------------------------------------------------
Flow ID: 1
From: 172.16.100.2:1023
To: 172.16.200.2:53
Protocol: UDP
-------------------------------------------------------
Duration                = 13.7721 s
Received packets        = 114
Minimum delay           = 0.039445 s
Maximum delay           = 11.4378 s
Average delay           = 6.88166 s
Average jitter          = 2.7429 s
Bytes received          = 22800
Average bitrate         = 13.2442 Kbit/s
Average packet rate     = 8.2776 pkt/s
Packets dropped         = 887
Percent dropped         = 88.6114 %
-------------------------------------------------------
// štatistiky dvoch tokov boli pre skrátenie výstupu  vynechané
-------------------------------------------------------
***************  TOTAL RESULTS   ******************
-------------------------------------------------------
Number of flows         = 3
Duration                = 13.7721 s
Received packets        = 828
Minimum delay           = 0.026793 s
Maximum delay           = 11.4378 s
Average delay           = 1.83279 s
Average jitter          = 0.656125 s
Bytes received          = 75420
Average bitrate         = 43.8103 Kbit/s
Average packet rate     = 60.1215 pkt/s
Packets dropped         = 1675
Percent dropped         = 66.9197 %
-------------------------------------------------------
```

Štatistika je rozdelená do dvoch sekcií. V prvej časti sú štatistiky jednotlivých tokov, druhá obsahuje celkové štatistiky za daný modul. Položky v štatistike jedného toku majú takýto význam:

- Flow ID – jednoznačný identifikátor toku
- From – zdrojová IP adresa a zdrojový port toku
- To – cieľová IP adresa a cieľový port toku
- Protocol – transportný protokol
- Duration – dĺžka trvania generovania toku (čas od vygenerovania prvého paketu až po čas prijatia posledného paketu)
- Received packets – počet paketov toku prijatý analyzátorom
- Minimum delay – čas minimálneho oneskorenia paketu daného toku
- Maximum delay – čas maximálneho oneskorenia paketu daného toku
- Average delay – priemerné oneskorenie paketov daného toku
- Bytes received – celkový počet bytov toku prijatých analyzátorom
- Average bitrate – priemerná prenosová rýchlosť toku učená z počtu prenesených bajtov a dĺžky trvania prenosu
- Average packet rate – priemerný počet prijatých paketov za sekundu
- Packets dropped – počet paketov, ktoré boli odoslané generátorom, ale neboli prijaté analyzátorom
- Percent dropped – percento stratených paketov

Položky celkových štatistík majú rovnaký význam, ako u jednotlivých tokov s tým rozdielom, že ide o súhrn informácií zo všetkých tokov, pre ktoré je daný modul v úlohe analyzátora.

Kapitola 6

Simulácia QoS pomocou implementovaných nástrojov

V prvej časti tejto kapitoly budú implementované moduly podrobené verifikácií voči reálnym zariadeniam. Na niekoľkých typových úlohách, kde bude totožná konfigurácia QoS nastavená na reálnych smerovačoch a v simulačnom prostredí, budú porovnávané parametre dátových prenosov generovaných softwarovým generátorom voči dátovým tokom generovaných simulačným modulom. Cieľom je zistenie, či následné simulovanie implementovanými modulmi je relevantné a zodpovedá skutočnosti.

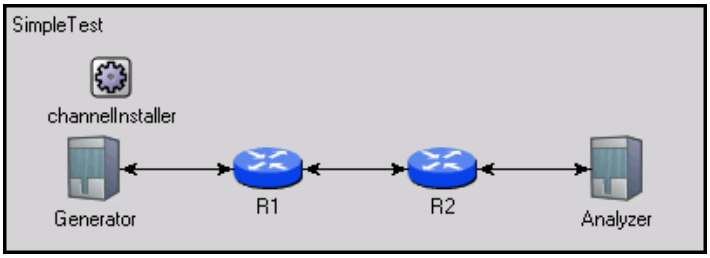
V druhej časti kapitoly budú na prípadovej štúdii prezentované možnosti simulácie QoS pomocou rozšírení implementovaných v rámci tejto práce. Prípadová štúdia bude okrem QoS modulov vyžívať aj modul dynamického smerovania OSPF a modul ScenarioManager, vďaka čomu bude možné sledovať dynamické zmeny v topológii. Bude ukázané ako nastavenia QoS môžu pomôcť adaptácii sieti na výpadok linky a zaručiť adekvátne prenosové parametre pre prioritné aplikácie.

Zdrojové súbory všetkých simulácií vykonaných v rámci tejto kapitoly sú dostupné na priloženom CD.

6.1 Verifikácia implementovaných modulov

Verifikácia bola realizovaná na jednoduchej topológii zapojenej podľa obrázku 6.1. Topológia pozostávala z dvoch smerovačov navzájom prepojených PPP (Point-to-point) linkou. K obom smerovačom bol pripojený vlastný LAN segment. LAN segment smerovača R1 obsahoval stanicu, ktorá bola použitá ku generovaniu dátových tokov a v LAN segmente smerovača R2 sa nachádzala stanica, ktorá bola cieľom a analyzátorom generovaných tokov. Smerovanie medzi jednotlivými LAN segmentmi bolo zaistené pomocou staticky nakonfigurovaných ciest. Záujmovým miestom pre konfiguráciu a sledovanie QoS bolo PPP rozhranie smerovača R1, kde vznikalo úzke hrdlo na trase prenášaných paketov.

Pre simuláciu bola topológia modelovaná pomocou modulov ANSARouter a ANSATrafGenHost. Na strane reálnych zariadení boli použité smerovače Cisco, konkrétne model ISR 2811 s verziou operačného systému IOS 15.1.4M4. Stanice v LAN segmentoch mali nainštalovaný operačný systém Ubuntu verzie 11.10 (32-bit) a ku generovaniu a analýze dátových tokov bol použitý software D-ITG verzie 2.8.0-rc1. Parametre nastaviteľné v rámci simulácie bolo možné 1:1 aplikovať aj na konfiguráciu smerovačov Cisco. Štatistiky analyzované a pozorovateľné pomocou implementovaného modulu TrafGen sa z väčšej časti zhodujú



Obr. 6.1: Testovacia topológia

s tými, ktoré sú dostupné v rámci softwarového generátoru D-ITG.
V nasledujúcej časti budú prezentované jednotlivé testové úlohy a získané výsledky.

6.1.1 Test 1

V rámci tohto testu boli po dobu 10 sekúnd generované 3 toky s parametrami uvedenými v tabuľke 6.1.

	ID 1	**ID 3**	**ID 2**
Zdrojová IP	172.16.100.2	172.16.100.2	172.16.100.2
Cieľová IP	172.16.200.2	172.16.200.2	172.16.200.2
DSCP	default	CS2	EF
Trans. protokol	UDP	UDP	UDP
Zdrojový port	1023	1580	16384
Cieľový port	53	2480	16384
Aplikácia	custom	custom	voice
Parametre apl.	Frekvencia – 100 pps Veľkosť – 200B	Frekvencia – 100 pps Veľkosť – 200B	Kódek – G.729 Frekvencia – 50 pps VAD – vypnutý

Tabuľka 6.1: Toky generované v teste 1

PPP linka medzi smerovačom R1 a R2 bola nastavená na prenosovú rýchlosť 64Kbps. Testovanie prebiehalo na všetkých štyroch typoch implementovaných front. Nastavenie parametrov u jednotlivých typov bolo nasledovné:
Nastavenie FIFO fronty:

- Dĺžka fronty 50 paketov

Nastavenie WFQ fronty:

- HoldQueueLength – 1000
- CDT – 64
- DynamicQueues – 256

Nastavenie CQ fronty:

- Queue 1 – klasifikácia na základe DSCP hodnoty EF, dĺžka 20 paketov, váha 3000 bajtov

- Queue 2 – implicitná fronta, dĺžka 60 paketov, váha 1500 bajtov
- Queue 3 – klasifikácia toku s ID 1 na základe ACL, dĺžka 30 paketov, váha 1500 bajtov

Nastavenie PQ fronty:

- HighQueue – klasifikácia na základe DSCP hodnoty EF, dĺžka 20 paketov
- MediumQueue – klasifikácia na základe IP precedence 2, dĺžka 30 paketov
- LowQueue – implicitná fronta, dĺžka 60 paketov

	Simulácia	**Reálne zariadenia**
	doba prenosu / prijaté pakety / stratené pakety / priemerná rýchlosť / priemerné oneskorenie / rozptyl oneskorenia	
Tok 1		
FIFO	10.9s / 242 / 759 / 35.5Kbps / 0.992s / 0.101s	11.1s / 208 / 792 / 29.9Kbps / 1.359s / 0.241s
WFQ	13.7s / 114 / 887 / 13.2Kbps / 6.881s / 2.743s	14.2s / 112 / 888 / 12.6Kbps / 8.193s / 2.696s
CQ	11.8s / 131 / 870 / 17.8Kbps / 2.502s / 0.717s	12.5s / 136 / 864 / 17.4Kbps / 2.735s / 0.924s
PQ	12.6s / 61 / 940 / 7.7Kbps / 11.307s / 1.133s	13.0s / 60 / 940 / 7.4Kbps / 11.536s / 0.369s
Tok 2		
FIFO	10.4s / 64 / 937 / 9.7Kbps / 0.906s / 0.253s	11.0s / 122 / 878 / 17.7Kbps / 1.288s / 0.222s
WFQ	12.4s / 213 / 788 / 27.3Kbps / 3.316s / 1.046s	12.3s / 221 / 779 / 28.7Kbps / 3.962 s / 0.964s
CQ	12.7s / 165 / 836 / 20.7Kbps / 4.154s / 1.533s	13.8s / 168 / 832 / 19.5Kbps / 5.082s / 2.010s
PQ	10.8s / 227 / 774 / 33.6Kbps / 1.392s / 0.289s	10.9s / 231 / 769 / 33.9Kbps / 1.560s / 0.299s
Tok 3		
FIFO	10.9s / 234 / 267 / 3.5Kbps / 0.999s / 0.059s	10.7s / 212 / 288 / 3.1Kbps / 1.202s / 0.111s
WFQ	10.0s / 501 / 0 / 8.0Kbps / 0.053s / 0.015s	10.1s / 500 / 0 / 8.0Kbps / 0.109s / 0.021s
CQ	10.0s / 487 / 14 / 7.9Kbps / 0.229s / 0.107s	10.1s / 479 / 21 / 7.6Kbps / 0.573s / 0.258s
PQ	10.0s / 501 / 0 / 8.0Kbps / 0.033s / 0.007s	10.0s / 501 / 0 / 8.0Kbps / 0.088s / 0.017s

Tabuľka 6.2: Namerané hodnoty testu 1

6.1.2 Test 2

V rámci tohto testu bolo po dobu 10 sekúnd generovaných 5 tokov s parametrami uvedenými v tabuľke 6.3.

PPP linka medzi smerovačom R1 a R2 bola nastavená na prenosovú rýchlosť 512Kbps. Testovanie prebiehalo na všetkých štyroch typoch implementovaných front s takýmito nastaveniami:
Nastavenie FIFO fronty:

- Dĺžka fronty 100 paketov

Nastavenie WFQ fronty:

- HoldQueueLength – 1000
- CDT – 10
- DynamicQueues – 256

	ID 1	**ID 3**	**ID 2**	**ID 4**	**ID 5**
Zdrojová IP	172.16.100.2	172.16.100.2	172.16.100.2	172.16.100.2	172.16.100.2
Cieľová IP	172.16.200.2	172.16.200.2	172.16.200.2	172.16.200.2	172.16.200.2
DSCP	EF	EF	default	default	default
Trans. protokol	UDP	UDP	UDP	UDP	TCP
Zdrojový port	16384	16385	16386	16387	1580
Cieľový port	16384	16385	16386	16387	2480
Aplikácia	voice	voice	voice	voice	custom
Parametre apl.	Kód.–G.711 Frekv.–50pps VAD–vyp.	Kód.–G.711 Frekv.–50pps VAD–vyp.	Kód.–G.711 Frekv.–50 pps VAD–vyp.	Kód.–G.711 Frekv.–50pps VAD–vyp.	Frekv.–100pps Veľk.–1000B

Tabuľka 6.3: Toky generované v teste 2

Nastavenie CQ fronty:

- Queue 1 – implicitná fronta, dĺžka 20 paketov, váha 1500 bajtov
- Queue 2 – klasifikácia na základe DSCP hodnoty EF, dĺžka 20 paketov, váha 1500 bajtov
- Queue 3 – klasifikácia toku s ID 4 na základe ACL, dĺžka 20 paketov, váha 1500 bajtov

Nastavenie PQ fronty:

- HighQueue – klasifikácia na základe DSCP hodnoty EF, dĺžka 20 paketov
- MediumQueue – klasifikácia toku s ID 5 na základe ACL, dĺžka 20 paketov
- LowQueue – implicitná fronta, dĺžka 100 paketov

	Simulácia	Reálne zariadenia
	doba prenosu / prijaté pakety / stratené pakety / priemerná rýchlosť / priemerné oneskorenie / rozptyl oneskorenia	
Tok 1		
FIFO	10.4s / 487 / 14 / 59.8Kbps / 0.228s / 0.198s	10.5s / 439 / 61 / 53.5Kbps / 0.381s / 0.180s
WFQ	10.0s / 501 / 0 / 64.1Kbps / 0.010s / 0.007s	10.0s / 500 / 0 / 64.0Kbps / 0.098s / 0.031s
CQ	10.0s / 501 / 0 / 64.1Kbps / 0.014s / 0.011s	10.1s / 500 / 0 / 63.4Kbps / 0.168s / 0.057s
PQ	10.0s / 501 / 0 / 64.1Kbps / 0.009s / 0.004s	10.0s / 500 / 0 / 64.0Kbps / 0.078s / 0.029s
Tok 2		
FIFO	10.4s / 455 / 46 / 55.9Kbps / 0.27s / 0.184s	10.6s / 479 / 21 / 58.4Kbps / 0.445s / 0.210s
WFQ	10.0s / 501 / 0 / 64.1Kbps / 0.013s / 0.007s	10.0s / 500 / 0 / 64.0Kbps / 0.096s / 0.040s
CQ	10.0s / 501 / 0 / 64.1Kbps / 0.019s / 0.011s	10.1s / 500 / 0 / 63.4Kbps / 0.147s / 0.056s
PQ	10.0s / 501 / 0 / 64.1Kbps / 0.011s / 0.004s	10.0s / 500 / 0 / 64.0Kbps / 0.075s / 0.022s
Tok 3		
FIFO	10.4s / 431 / 70 / 52.9Kbps / 0.252s / 0.174s	10.5s / 444 / 56 / 54.1Kbps / 0.350s / 0.177s
WFQ	10.0s / 501 / 0 / 64.1Kbps / 0.020s / 0.011s	10.1s / 500 / 0 / 63.5Kbps / 0.131s / 0.043s
CQ	10.0s / 474 / 27 / 60.7Kbps / 0.098s / 0.113s	10.2s / 461 / 39 / 57.9Kbps / 0.227s / 0.140s
PQ	23.8s / 389 / 112 / 20.9Kbps / 2.265s / 2.206s	23.1s / 369 / 131 / 20.4Kbps / 2.645s / 2.114s
Tok 4		
FIFO	10.4s / 427 / 74 / 52.4Kbps / 0.252s / 0.172s	10.5s / 458 / 42 / 55.8Kbps / 0.401s / 0.198s
WFQ	10.0s / 501 / 0 / 64.1Kbps / 0.023s / 0.011s	10.1s / 500 / 0 / 63.5Kbps / 0.117s / 0.050s
CQ	10.0s / 501 / 0 / 64.1Kbps / 0.019s / 0.012s	10.1s / 500 / 0 / 63.4Kbps / 0.162s / 0.051s
PQ	22.2s / 388 / 113 / 22.3Kbps / 2.277s / 2.211s	23.0s / 372 / 128 / 20.7Kbps / 2.578s / 2.020s
Tok 5		
FIFO	23.3s / 1001 / 71 / 342.4Kbps / 9.506s / 3.187s	22.2s / 1000 / – / 360.4Kbps / – / –
WFQ	24.9s / 1001 / 38 / 321.0Kbps / 11.54s / 2.553s	27.5s / 1000 / – / 290.9Kbps / – / –
CQ	25.4s / 1001 / 43 / 315.0Kbps / 11.9s / 2.807s	24.8s / 1000 /– / 342.4Kbps / – / –
PQ	23.4s / 1001 / 61 / 341.9Kbps / 9.85s / 2.783s	22.6s / 1000 / – / 353.9Kbps / – / –

Tabuľka 6.4: Namerané hodnoty testu 2

6.1.3 Zhodnotenie výsledkov testov

Pri bližšom pohľade na hodnoty získané v rámci testov zo sekcií 6.1.1 a 6.1.2 je možné pozorovať, že počet prijatých, respektíve stratených paketov pri použití front WFQ, CQ a PQ sa rádovo zhoduje. Keď to vyjadríme percentuálne, tak hodnota odchýlky stratovosti pre WFQ je 0,22%, pre CQ 0,62% a pre PQ 0,86%. Štatisticky to sú prakticky zanedbateľné hodnoty. Horší je už výsledok pri použití FIFO fronty, tu sa odchýlka stratovosti rovná 5,13%. Treba však poznamenať, že tento rozdiel nie je spôsobený chybnou implementáciou FIFO fronty, ale rôznym spôsobom generovania paketov. V rámci simulácie sú pakety generované vždy v rovnakom poradí bez ohľadu na počet spustení toho istého testu. Softwarový generátor D-ITG pravdepodobne jednotlivé toky generuje v samostatných vláknach a pri preemtívnom plánovaní nemusí byť zaručené rovnaké poradie paketov v prípade, že v tom istom čase má dôjsť ku generovaniu paketu dvoch alebo viacerých tokov. Aj kvôli tomuto správaniu boli testy na reálnych zariadeniach realizované trikrát a za celkový výsledok sa zobral priemer hodnôt všetkých iterácií. Každopádne výsledky pri použití FIFO fronty sú z veľkej časti závislé na poradí, v akom sa pakety dostanú do fronty a v prípade týchto testov poradie nebolo rovnaké, teda aj výsledky sa od seba líšia.

Druhou veličinou, ktorú môžeme v rámci výsledkov pozorovať je čas. Ide najmä o hodnoty oneskorenia a rozptylu oneskorenia. V tomto prípade už výsledky nie sú až tak uspo-

kojivé. Odchýlky sa pohybujú v rozmedzí 0,010s až 1,312s, čo nie sú zanedbateľné hodnoty. Príčinou sú rozdiely pri zarátavaní rôznych typov oneskorenia v rámci simulácie respektíve pri testovaní v reálnom prostredí. Kým simulácia do celkového oneskorenia počíta iba serializačné oneskorenie, propagačné oneskorenie (staticky konfigurovateľné pri vytváraní simulácie) a oneskorenie spôsobené radením do front, v reálnom prostredí sa k týmto trom typom pridáva oneskorenie spôsobené spracovaním na zariadeniach. Práve veľká variabilita tohto štvrtého typu oneskorenia je príčinou značných odlišností. Ďalším problémom je aj použitý softwarový generátor D-ITG, ktorý nemá implicitnú synchronizáciu času a k správnej funkcii potrebuje externý zdroj času. Nedokonalá synchronizácia času je ďalšou príčinou odchýlok.

Osobitným prípadom je porovnanie TCP tokov. Z pohľadu času nie je možné relevantne, okrem celkovej doby prenosu, porovnať tento typ prenosu. Zatiaľ čo simulácia vyhodnocuje jednosmerné oneskorenia, D-ITG počíta čas od odoslania paketu až po prijatie potvrdenia o prenose tzv. round-trip time. Taktiež pomocou softwaru D-ITG nebolo možné pre TCP prenosy určovať stratovosť. Z týchto dôvodov nebol na TCP prenosy pri testovaní kladený väčší dôraz. Z celkového pohľadu sa simulácia pomocou vytvorených modulov dá považovať za relevantnú, najmä z pohľadu stratovosti paketov. Z pohľadu oneskorenia je potom vhodné sledovať časové rozdiely v rámci simulácie za použitia rôznych typov front. Získané časové údaje však určite nemožno brať ako absolútnu hodnotu.

6.2 Prípadová štúdia

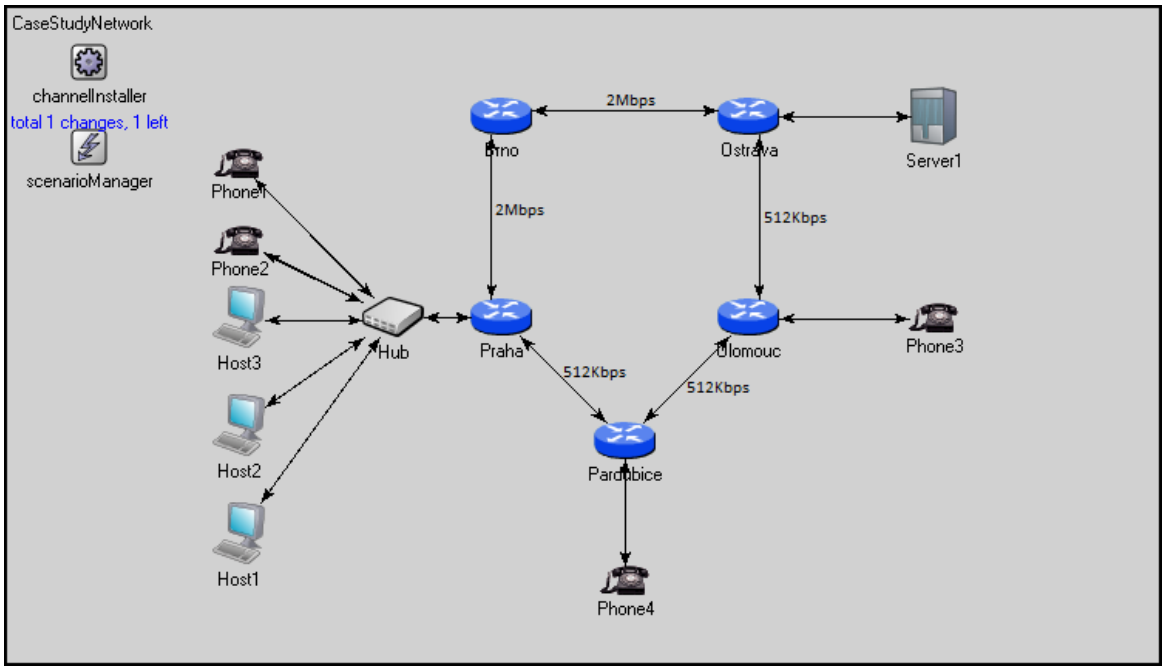
Cieľom tejto prípadovej štúdie je priblížiť možnosti simulácie QoS pomocou modulov vytvorených v rámci tejto práce. Prípadová štúdia simuluje jednoduchú firemnú sieť s pobočkami prepojenými WAN sieťou. Použitá topológia siete, zobrazená na obrázku 6.2, pozostáva z piatich lokalít (Praha, Brno, Ostrava, Olomouc, Pardubice). Lokality sú medzi sebou prepojené do kruhu s tým, že trasa Praha–Brno–Ostrava má rýchlosť 2Mbps a trasa Praha–Pardubice–Olomouc–Ostrava dosahuje rýchlosť 512Kbps. Smerovače si medzi sebou vymieňajú smerovacie informácie pomocou protokolu OSPF. Topológia je redundantná a odolná voči výpadku jednej linky. Po skonvergovaní smerovacieho protokolu na základe metrík liniek, je z pohľadu smerovača v lokalite Praha realizovaný prenos po trase Praha–Brno–Ostrava–Olomouc. Pre prenos Praha–Padubice je použitá priamo pripojená linka.

V prípadovej štúdii je simulovaná situácia, kedy klientske stanice (Host 1–3) v lokalite Praha komunikujú so serverom (Server1) v dátovom centre lokality Ostrava. Popis parametrov jednotlivých tokov je uvedený v tabuľke 6.5.

Popis	**Zdroj**	**Cieľ**	**Trasport. protokol**	**Paketov za sek.**	**Časové rozloženie**	**Veľkosť paketu**	**Rozloženie veľkosti**
HTTP 5	Server1	Host1	TCP	80	exponenciálne	1000	gausovské
HTTP 6	Server1	Host2	TCP	50	exponenciálne	500	gausovské
HTTP 7	Server1	Host3	TCP	50	exponenciálne	500	gausovské
TFTP 8	Server1	Host3	UDP	200	konštantné	30	konštantné

Tabuľka 6.5: Parametre dátových tokov

Okrem dátových tokov medzi serverom a stanicami je simulovaná VoIP komunikácia medzi lokalitami Praha-Olomouc a Praha-Pardubice. Parametre komunikácie sú uvedené v tabuľke 6.6.



Obr. 6.2: Topológia prípadovej štúdie

Popis	**Participant 1**	**Participant 2**	**Kódek**	**Paketov za sek.**	**VAD**
VoIP 2	Phone1	Phone3	G.711	50	vypnutý
VoIP 4	Phone2	Phone4	G.711	50	vypnutý

Tabuľka 6.6: Parametre VoIP tokov

Hlasové pakety sú značkované DSCP hodnotou EF. Ostatné pakety sú generované s implicitnou DSCP hodnotou 0. Smer jednotlivých tokov je graficky znázornený na obrázku 6.3. Doba generovania je 100 sekúnd.

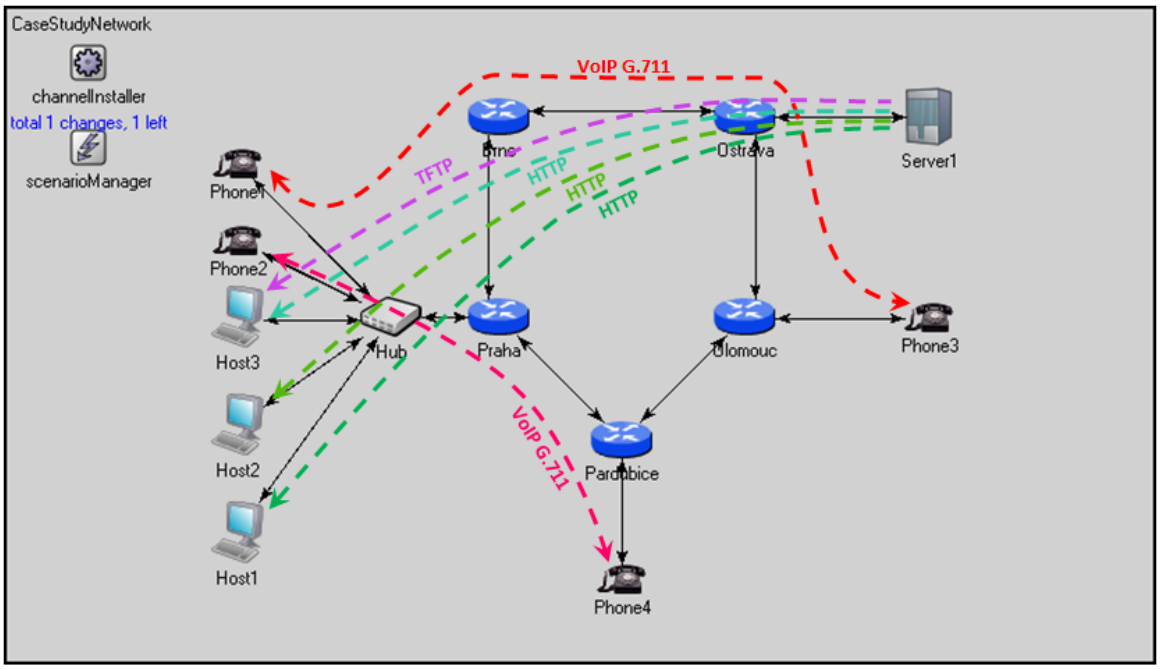
V nasledujúcej časti sú diskutované rôzne scenáre simulované nad definovanou topológiou.

6.2.1 Implicitné nastavenie QoS bez výpadku linky

V tomto scenári bola konfigurácia QoS ponechaná v implicitnom nastavení. Teda na všetkých WAN linkách bola FIFO fronta o dĺžke 75 paketov. Simulácia neobsahovala žiadny výpadok linky. Výsledky prenosov, ako ich zaznamenal modul TrafGen, sú uvedené v tabuľke 6.7.

Obrázok 6.4 zobrazuje priebeh dĺžky výstupnej fronty smerovača v lokalite Ostrava na rozhraní linky k smerovaču v lokalite Brno. Ide o frontu, ktorá je z pohľadu analýzy najpodstatnejšia, keďže je tam úzke hrdlo pri prestupe paketov z LAN segmentu dátového centra do WAN.

Z pozorovania výsledkov je vidieť, že sieť kapacitne postačuje k prenosu daného množstva dát a nedochádza k výrazným stratám paketov. K zahlteniu na výstupných frontách



Obr. 6.3: Toky simulované v prípadovej štúdii

Popis	**Doba prenosu**	**Prijaté pakety**	**Stratené pakety**	**Priemerná rýchlosť**	**Priemerné oneskorenie**	**Rozptyl oneskorenia**
HTTP 5	100s	7817	9	629.4 Kbps	0.056s	0.014s
HTTP 6	100s	4935	7	198.1 Kbps	0.065s	0.019s
HTTP 7	100s	4845	9	195.8 Kbps	0.064s	0.019s
TFTP 8	100s	20001	0	479.9 Kbps	0.036s	0.011s
VoIP 2	100s	5001	0	64.0 Kbps	0.048s	0.011s
VoIP 4	100s	5001	0	64.0 Kbps	0.013s	0.000003s

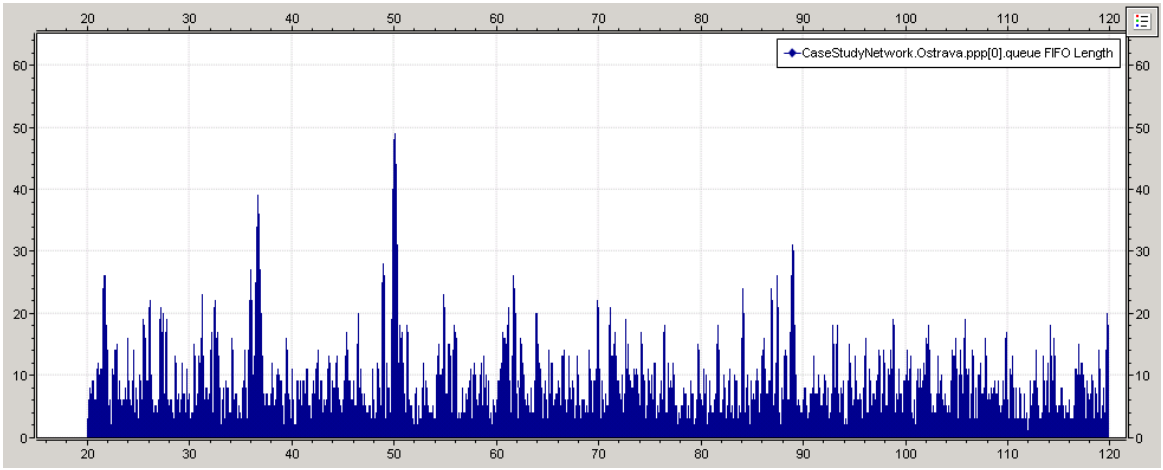
Tabuľka 6.7: Štatistiky simulovaných tokov

nedochádza a kvalita prenosu služby VoIP je v tomto prípade vyhovujúca aj bez špeciálnych nastavení QoS.

6.2.2 Implicitné nastavenie QoS s výpadkom linky

Počas simulácie v tomto scenári boli nastavenia QoS ponechané na FIFO s dĺžkou fronty 75 paketov, ale počas simulácie, 20 sekúnd pred koncom generovania paketov (simulačný čas 100), došlo k zámernému výpadku linky Praha–Brno. Po skonvergovaní protokolu OSPF prenos pokračoval ďalej po záložnej linke (Ostrava–Olomouc–Pardubice–Praha). Tabuľka 6.8 zachytáva výsledné štatistiky sledovaných tokov.

Zo zaznamenaných výsledkov je zrejmé, že po výpadku dochádza k preťaženiu siete a týmto preťažením trpia aj kritické VoIP aplikácie. Dochádza k vysokej stratovosti paketov a na kritickú hodnotu sa dostáva aj oneskorenie a rozptyl oneskorenia. Prakticky hlasové služby sú po výpadku nepoužiteľné. TCP aplikáciam sa doba prenosu predĺžila skoro o 50%.



Obr. 6.4: Priebeh dĺžky front na WAN linkách smerovača v lokalite Ostrava

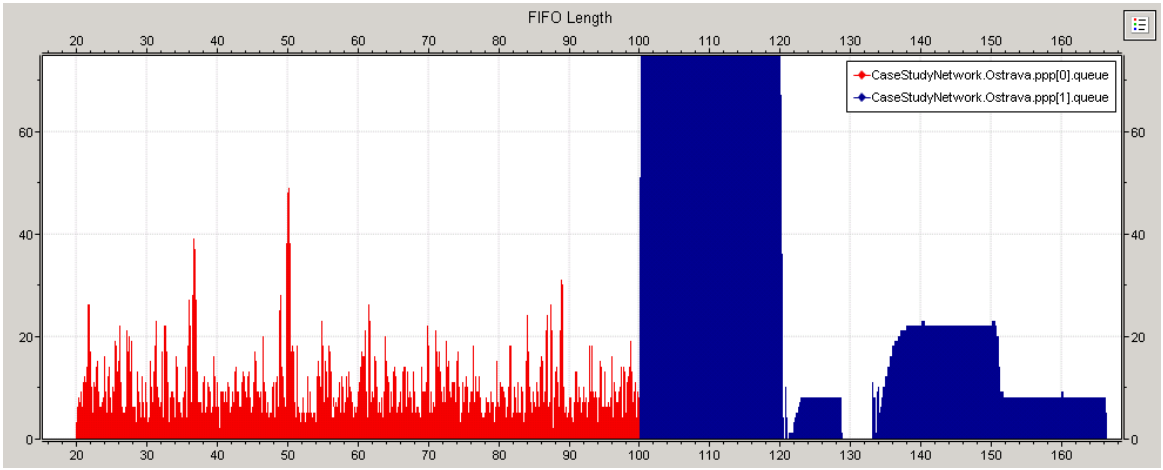
Popis	**Doba prenosu**	**Prijaté pakety**	**Stratené pakety**	**Priemerná rýchlosť**	**Priemerné oneskorenie**	**Rozptyl oneskorenia**
HTTP 5	146.5s	7812	73	426.8 Kbps	8.608s	7.685s
HTTP 6	108.8s	4926	117	182.3 Kbps	2.954s	2.566s
HTTP 7	131.3s	4889	107	195.8 Kbps	6.885s	6.058s
TFTP 8	101.1s	19016	985	451.3 Kbps	0.210s	0.168s
VoIP 2	100.7s	4643	358	59.0 Kbps	0.137s	0.091s
VoIP 4	100.3s	4730	271	60.3 Kbps	0.062s	0.045s

Tabuľka 6.8: Štatistiky simulovaných tokov

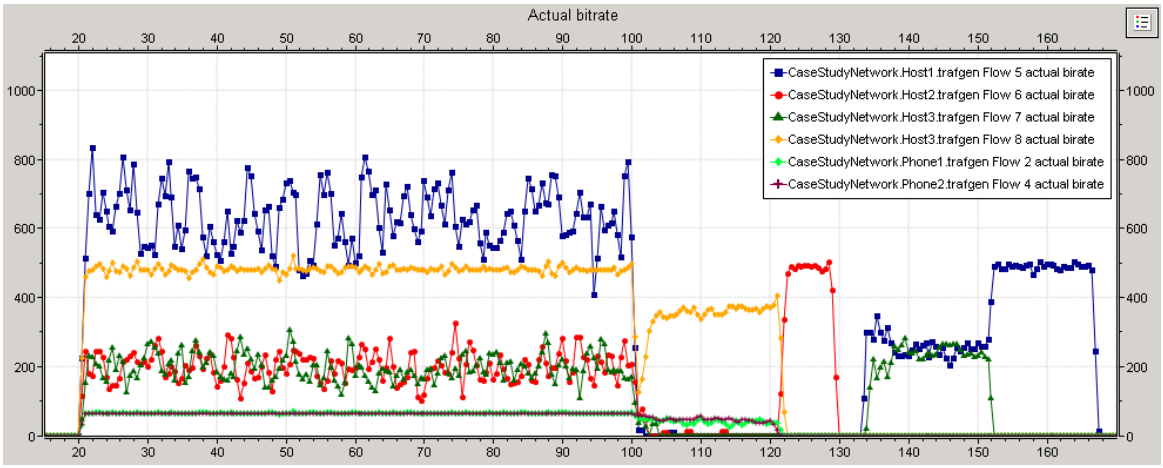
Keď sa pozrieme na graf priebežnej dĺžky front smerovača v lokalite Ostrava (obrázok 6.5), tak je vidieť, že do doby výpadku je fronta na linke k smerovaču Brno využívaná podobne ako v predchádzajúcom prípade. Po výpadku však okamžite dochádza k zaplneniu fronty rozhrania k smerovaču Olomouc. Fronta si drží plnú kapacitu až do doby, kým je skončené generovanie UDP paketov toku TFTP 8. Následne už fronta dosahuje priemernú dĺžku približne 15 paketov, keďže dochádza k prenosu len TCP aplikácií, ktoré dokážu adaptovať svoju prenosovú rýchlosť.

Z pohľadu analýzy je tiež zaujímavé sledovať vývoj aktuálnej prenosovej rýchlosti, ako ju vníma cieľ daného toku. Obrázok 6.6 pravé zachytáva graf aktuálnych prenosových rýchlostí jednotlivých tokov počas simulácie tohto scenára. Do výpadku sú prenosy stabilné (pre TCP je typické, že prenosová rýchlosť miene kolíše a adaptuje sa aktuálnemu zahlteniu siete). Po výpadku je pozorovateľná okamžitá zmena. Fronta sa zaplní a TCP toky sú úplne zablokované UDP tokmi. Dôvodom je to, že UDP toky nereagujú na zahlnenie siete a pakety sú zasielané rovnakou rýchlosťou bez ohľadu na stratovosť (spomalenie musí byť u UDP tokov implementované na aplikačnej úrovni). Po dokončení UDP prenosov sa postupne obnovujú aj TCP prenosy a sú úspešne dokončené. V reálmom prenose je toto dokončenie podmienené časovým limitom aplikačnej vrstvy, ktorá môže spojenie prehlásiť za nefunkčné.

Na tomto scenári jasne vidieť ako výpadok v sieti s redundanciou môže prakticky zničiť funkčnosť a dosupnosť všetkých aplikácií. Je to čiastočné spôsobené nedostatočnou kapaci-



Obr. 6.5: Priebeh dĺžky front na WAN linkách smerovača v lokalite Ostrava



Obr. 6.6: Priebeh aktuálnych prenosových rýchlostí sledovaných tokov

tou záložných liniek, ale hlavnú vinu v tomto prípade nesie nedostatočná ochrana aplikácii na úrovni nastavenia QoS.

6.2.3 WFQ s výpadkom linky

V predchádzajúcom scenári boli ponechané implicitné nastavenia QoS a výsledky po výpadku boli z pohľadu kvality prenosu neakceptovateľné. V tomto scenári bol opäť simulovaný výpadok linky, rozdielom bolo nastavenie WFQ fronty na všetky WAN rozhrania. Nastavené parametre WFQ boli nasledovné:

- HoldQueueLength – 1000
- CDT – 64

- DynamicQueues – 256

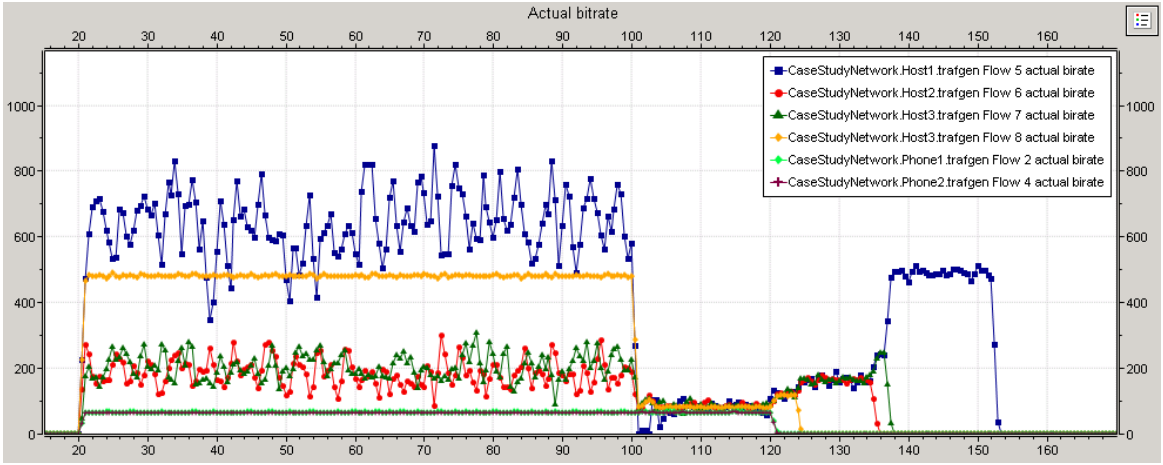
Výsledky prenosov ako ich zaznamenal modul TrafGen sú uvedené v tabuľke 6.9.

Popis	**Doba prenosu**	**Prijaté pakety**	**Stratené pakety**	**Priemerná rýchlosť**	**Priemerné oneskorenie**	**Rozptyl oneskorenia**
HTTP 5	132.0s	7891	23	474.9 Kbps	5.239s	4.723s
HTTP 6	114.6s	4712	7	163.1 Kbps	2.226s	1.994s
HTTP 7	116.6s	5136	12	177.1 Kbps	2.513s	2.256s
TFTP 8	103.5s	16835	3166	390.1 Kbps	0.185s	0.156s
VoIP 2	100.0s	5001	0	63.9 Kbps	0.043s	0.005s
VoIP 4	100.3s	5001	0	63.9 Kbps	0.016s	0.002s

Tabuľka 6.9: Štatistiky simulovaných tokov

V tomto prípade sú hodnoty zachytené u VoIP komunikácie úplne vyhovujúce bez straty jediného paketu a nízkym oneskorením. Je to spôsobené nastaveným značkovaním DSCP, vďaka ktorému WFQ preferuje VoIP pakety.

To, že WFQ je schopná zabezpečiť rovnomerné rozdelenie šírky pásma je zrejmé aj z grafu aktuálnych prenosových rýchlostí 6.7. V tomto prípade dátový tok TFTP 8 bol obmedzený a dostalo sa mu rovnaké množstvo prostriedkov ako jednotlivým TCP tokom. Postupne ako dochádza k ukončeniu prenosu niektorých z tokov sa šírka pásma navyšuje pre ostávajúce aktívne toky.



Obr. 6.7: Priebeh aktuálnych prenosových rýchlostí sledovaných tokov

Simulácia ukázala, že zmenou nastavenia QoS na linkách je možné docieliť zachovanie funkčnosti aplikácií aj prípade výpadku primárnej linky. Nedochádza k nerovnomernému rozdeleniu prostriedkov, ako je to v prípade FIFO fronty. TCP aplikácie dostanú svoju časť pásma a nie sú obmendzené UDP aplikáciami s veľkými nárokmi na prenosové pásmo. Toto je aj dôvod prečo výrobcovia smerovačov odporúčajú na linkách so šírkou pásma nižšou ako 768Kbps použiť práve WFQ fronty.

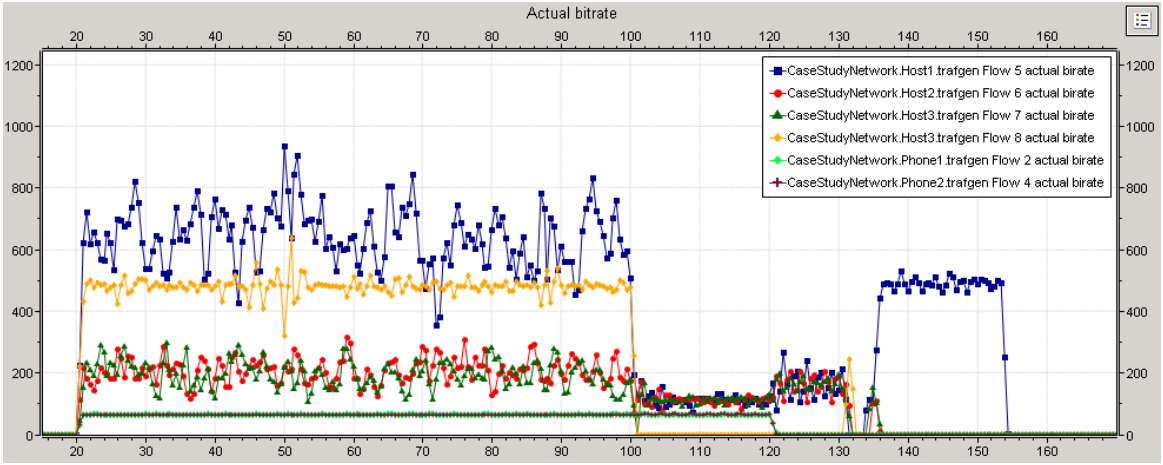
6.2.4 PQ s výpadkom linky

Problémom scenára s FIFO frontou bolo, že UDP tok s veľkými nárokmi na šírku pásma kompletne zahltil sieť. WFQ fronta tento problém vyriešila, ale jej mechanizmus nedokáže zabezpečiť striktnú prioritu pre kritické aplikácie. V tomto scenári bola použitá PQ fronta, v ktorej VoIP pakety boli mapované do High podfronty, TCP pakety do Medium podfronty a pakety problémového toku TFTP 8 implicitne do Normal podfronty.
Výsledné štatistiky prenosov zachytáva tabuľka 6.10.

Popis	**Doba prenosu**	**Prijaté pakety**	**Stratené pakety**	**Priemerná rýchlosť**	**Priemerné oneskorenie**	**Rozptyl oneskorenia**
HTTP 5	133.4s	7912	37	481.0 Kbps	5.365s	4.839s
HTTP 6	114.9s	5031	60	179.9 Kbps	1.521s	1.363s
HTTP 7	115.1s	4879	64	171.0 Kbps	1.519s	1.368s
TFTP 8	111.2s	16094	3907	347.0 Kbps	0.241s	0.216s
VoIP 2	100.0s	5001	0	64.0 Kbps	0.039s	0.003s
VoIP 4	100.3s	5001	0	64.0 Kbps	0.015s	0.001s

Tabuľka 6.10: Štatistiky simulovaných tokov

Kvalita VoIP sa mierne vylepšila, čo sa dalo očakávať. V tomto prípade sa však situácia medzi TCP a UDP tokmi obrátila. Po výpadku sa pakety toku TFTP 8 prestali dostávať k ich cieľu a celá dostupná šírka pásma bola rozdelená medzi HTTP prenosy. Spôsobené je to tým, že vždy sa obsluhuje fronta s vyššou prioritou a u front s nižšou prioritou dochádza k efektu vyhladovania.



Obr. 6.8: Priebeh aktuálnych prenosových rýchlostí sledovaných tokov

Simulácia ukázala, že PQ fronta vôbec nepriniesla žiadne vylepšenie, iba problém presunula na iné miesto. Teda ako najlepšia možnosť sa ukazuje použitie fronty WFQ. Niektorí výrobcovia smerovačov však majú vo svojich zariadeniach implementované vlastné algoritmy, ktoré vedia spojiť výhody WFQ a PQ. Takýmto algoritmom je napríklad LLQ (Low Latency Queuing) od spoločnosti Cisco, ktorý by bol pri reálnom nasadení na topológiu tejto prípadovej štúdie preferovaným riešením. Pre VoIP pakety by zabezpečil striktnú

prioritu a zvyšnú šírku pásma by mohol spravodlivo rozdeliť medzi aktívne toky. Princíp tohto algoritmu je však spoločnosťou Cisco utajovaný.

Kapitola 7

Záver

Táto práca sa zaoberá simuláciou a modelovaním QoS (Quality of Service) v počítačových sieťach pomocou simulátora OMNET++.

V kapitole 2 bol podaný ucelený pohľad na nástroje, ktorými je možné zabezpečiť QoS v IP sieťach. Konkrétne sa zameriava na model diferencovaných služieb. Analyzuje spôsob klasifikácie a značkovania paketov. Podrobne predstavuje rôzne typy front, ktorými je možné spravovať zahltenie na výstupných rozhraniach sieťových zariadení.

Kapitola 3 bola venovaná predstaveniu architektúry simulátora OMNeT++, jeho inštalácie, spôsobu modelovania a vytváraniu modulov. Vo zvyšnej časti kapitoly bol diskutovaný INET Framework ako nadstavba simulátora OMNeT++ pre simuláciu počítačových sietí.

V kapitole 4 bol dôkladne zanalyzovaný aktuálny stav a možnosti použitia modulov INET Framework pre simuláciu QoS. Boli rozobraté tri existujúce moduly – DropTailQueue, DropTailQosQueue a REDQueue. Vďaka týmto modulom bolo možné v rámci pôvodného INET Framework simulovať základne QoS funkcie.

Z analýzy modulov INET Framework v kapitole 4 vyplynulo, že možnosti simulácie QoS nie sú dostatočné a preto v rámci kapitoly 5 boli predstavené rozšírenia, ktoré umožňujú simuláciu pokročilých QoS funkcií. Konkrétne sa jedná o štyri typy výstupných front – FIFO(Fist In First Out), WFQ (Weighted Fair Queue), PQ (Priority Queue) a CQ (Custom Queue). Všetky tieto typy front boli implementované v module s názvom ANSAQosSystem, ktorý je možné použiť ako modul výstupnej fronty v ktoromkoľvek module fyzického rozhrania (napr. PPPInterface alebo EthernetInterface). V kapitole 5 je taktiež popísaná implementácia modulu s názvom TrafGen. Modul bol vytvorený z dôvodu nutnosti generovania dátových tokov, na ktorých prenose by bolo možné sledovať, ako rôznorodé nastavenia QoS ovplyvňujú prenosové parametre týchto tokov. Modul umožňuje generovanie TCP a UDP paketov a modulárnu definíciu aplikácií.

V poslednej kapitole 6 boli implementované moduly podrobené verifikácii voči reálnym zariadeniam. Testy je možné považovať za úspešné, lebo po stránke stratovosti paketov simulácia rámcovo zodpovedá skutočnosti. Z pohľadu času však hodnoty nemožno považovať za absolútne, a preto sa nedajú priamo zrovnávať s reálmi hodnotami. Tento problém nesúvisí iba so simuláciou QoS, ale prakticky všetkých sieťových záležitostí ako takých, keďže v simulačnom prostredí INET Framework absentuje akýkoľvek model procesora na sieťovom zariadení. Teda každá operácia spracovávaná v simulácii smerovačom má dĺžku trvania nula sekúnd simulačného času. Následkom tohto nedostatku je skreslené oneskorenie paketu pri prechode simulovanou trasou.
Vo zvyšku kapitoly sú na prípadovej štúdii prezentované možnosti simulácie QoS pomocou implementovaných modulov. Určite zaujímavé je graficky sledovať priebeh dĺžky jednotli-

vých front alebo priebeh aktuálnej rýchlosti konkrétneho dátového toku.

QoS je v súčasnej dobe veľmi obľúbená téma aj vďaka tomu, že v dnešných sieťach sa vyskytuje čoraz väčší počet konvergovaných služieb. Určite pre dizajnérov rozsiahlych sietí by bolo užitočné, pred samotnou implementáciu QoS v reálnej sieti, mať možnosť si odsimulovať, či ich navrhované riešenie spĺňa požiadavky a nároky prenášaných aplikácií. Výsledkom tejto práce je priblíženie možností simulácie QoS v prostredí INET Framework bližšie k realite. Bolo vytvorené prostredie, kde je možné podrobne analyzovať jednotlivé QoS nastavania a zisťovať ich dopad na výsledné správanie siete. V budúcnosti sa tak môže stať užitočným nástrojom pri návrhu QoS politík alebo pre edukačné účely.

Z pohľadu ďalšieho rozvoja by určite bolo vhodné implementovať algoritmy, ktoré je možné nájsť na moderných hardwarových smerovačoch ako napríklad CBWFQ (Class–Based Weighted Fair Queueing) alebo LLQ (Low Latency Queuing). Tieto algoritmy sú však proprietárne a ich definícia nie je zverejnená. Taktiež zaujímavými by boli implementácie funkcií regulácie a tvarovania prenosovej rýchlosti.

Na prácu by bolo možné nadviazať aj v zmysle ďalšieho experimentovania s modelmi a porovnávania ich relevancie s reálnym prostredím. Bolo by potrebné namerať dostatok údajov, vyhodnotiť ich a následne doplniť chýbajúce modely. Rovnako by bolo vhodné zaviesť metódu pre kompenzáciu nájdených nepresností modelov.

Literatúra

[1] Dokumentácia: INET Framework for OMNeT++/OMNEST. `http://inet.omnetpp.org/doc/INET/neddoc/index.html`.

[2] Odom, W.; Cavanaugh, M. J.: *Cisco QOS Exam Certification Guide, Second Edition.* Cisco Press, 2005, ISBN 1-58720-124-0.

[3] Odom, W.; Healy, R.; Donohue, D.: *CCIE Routing and Switching Certification Guide, Fourth Edition.* Cisco Press, 2010, ISBN 1-58705-980-0.

[4] Ranjbar, A. S.: *CCNP ONT Official Exam Certification Guide.* Cisco Press, 2007, ISBN 1-58720-176-3.

[5] Suchomel, T.: *Rozšíření simulátoru OMNeT++ o filtrovací pravidla ACL.* FIT VUT v Brně, 2009.

[6] Szigeti, T.: *End-to-End QoS Network Design.* Cisco Press, 2004, ISBN 1-58705-176-1.

[7] Varga, A.: *OMNeT++ Discrete Event Simulation System v4.0 - User Manual.* 2009.

[8] WWW stránky: Projekt ANSA. `https://nes.fit.vutbr.cz/ansa/`.

Dodatok A

Obsah CD

Umiestnenie na CD	**Popis**
examples/MyQosCaseStudy/	simulácia použitá v sekcii 6.2
examples/QosTest1/	simulácia použitá v sekcii 6.1.1
examples/QosTest2/	simulácia použitá v sekcii 6.1.2
install/ANSASources.zip	zdrojove súbory projektu ANSA
install/INETMANET-20080920.tbz2	inštalačné súbory INET Framework
install/omnetpp-4.0rc1.zip	inštalačné súbory OMNeT++
src/acl/	zdrojove súbory modulu AclContainer
src/qos/	zdrojove súbory modulu AnsaQosSystem
src/TrafficGenerator/	zdrojove súbory modulu TrafGen
dp-xdanko00.pdf	dokument technickej správy
dp-xdanko00.zip	zdrojove súbory technickej správy
readme.txt	návod na inštaláciu súborov z CD

Tabuľka A.1: Obsah CD